INFERENCE RULES AND PROOF PROCEDURES FOR INEQUATIONS

CHILUKURI K. MOHAN,* MANDAYAM K. SRIVAS,*† AND DEEPAK KAPUR‡

▷ The negation of equality is an important relation that arises naturally in the study of equational programming languages and logic programming with equality. Proving and solving equations and *inequations* may also constitute subtasks in constraint logic programming. In this paper, we give *forward* (i.e., nonrefutational) techniques for proving the negation of equality in a theory. We develop a complete inference system to check whether an inequation is a logical consequence of a given system of equations and inequations. The inference system is used to develop a goal-directed semidecision procedure which uses a narrowing technique for proving inequations. A decision procedure is obtained when certain additional conditions are satisfied. The semidecision procedure for proving inequations is also modified to obtain a semidecision procedure for solving inequations in a theory, i.e., finding a substitution such that the corresponding instance of the given inequation is a logical consequence of the given system. ◁

1. INTRODUCTION

The negation of equality is an important relation that arises naturally in the study of equational programming languages and logic programming with equality. Proving and solving equations and *inequations* may also constitute subtasks in con-

Address correspondence to Dr. C. K. Mohan, School of Computer and Information Science, Syracuse University, Syracuse, NY 13244.

Received May 1987; accepted August 1988.

*Research supported in part by the National Science Foundation Grant MCS8401624.

†Odyssey Research Associates, 301A Harris B. Dates Drive, Ithaca, NY 14850.

‡Research supported in part by the National Science Foundation Grant CCR8408461. Current address: Computer Science Dept., SUNY at Albany, NY 12222.

655 Avenue of the Americas, New York, NY 10010 0743-1066/90/$3.50

straint logic programming. In this paper we study equational reasoning in the presence of inequations, i.e., negation of equations with variables universally quantified.

1.1. Motivation

Equational programming languages constitute an important class of logic programming languages. Several equational languages and their implementations using the techniques of term rewriting have been proposed [7–9, 11, 13] in recent years. Unlike languages like PROLOG in which "equality" refers to the syntactical identity relation, equational languages reason with full equality in a theory defined by the program constituents. In the underlying logic of an equational program, " = " is an equivalence relation that satisfies the substitution property.

Declaratively, an equational program consists of a set of equational axioms, in which every variable is assumed to be universally quantified. Operationally, an equational program is viewed as a set of oriented equations (rules) in which "execution" proceeds as follows: any term matching with the *lhs* of an equation can be replaced by the corresponding *rhs*. Typical problems solved using an equational program with this simple operational execution mechanism are: (a) to find a "simplest" term equal to a given term, and (b) to check whether two given terms are equal.

The expressive power of equational programs is naturally extended with the if-then-else operator, which is useful in the concise and elegant axiomatic specifications of data types and functions [12]. For example, the *fetch* operation on an array data type can be specified by the equation

(0) $fetch(assign(a, i, x), j) =$ **(if** $i = j$ **then** x **else** $fetch(a, j)$**)**,

or equivalently by the conditional equations

(1) $i = j \Rightarrow fetch(assign(a, i, x), j) = x$,

(2) $i \neq j \Rightarrow fetch(assign(a, i, x), j) = fetch(a, j)$.

The inequation $i \neq j$ occurs explicitly in (2), and is implied in the **else** part of (0). It is necessary to understand reasoning in systems of equations and such inequations (negations of equations with variables universally quantified, e.g., $\forall i, j.\ i \neq j$) before conditional equations with inequations are explored. The need for using inequations arises naturally when we attempt to build complete data type and function specifications and reason with them. Sometimes, it is necessary to make use of certain unstated inequational assumptions such as the free constructor assumption [e.g., $nil \neq cons\ (x, y)$ is often used in reasoning about list specifications]. In this paper, we study reasoning in systems which consist of equations and inequations with universally quantified variables; some of the results given here were presented in [25], and are also contained in [26].

Equational reasoning in the presence of inequations is also relevant to the recently proposed "constraint logic programming" class of languages (CLP) [16]. CLP attempts to incorporate procedures for solving special kinds of constraints into a logic programming framework in a useful way. Several constraint-solving techniques have been investigated recently in the context of CLP [1, 17, 21–23]. While some of these works do consider negative constraints, they are applicable

either in the empty theory (i.e., syntactic unifiability and nonunifiability) or for a specific theory, such as linear arithmetic. It would be useful and important to extend constraints to those that require determining whether or not (instances of) two terms are equal in a given finitely specified equational theory supplemented with a set of inequations. Equational reasoning is also likely to play an important role in determining canonical forms for constraints in constraint logic programming as discussed in [23].

1.2. Related Work

Reasoning with the positive " $=$ " relation in equational systems is fairly well understood: the first completeness result for equational reasoning was obtained by Birkhoff [2]. Such is not the case when reasoning with the " $\neq$ " relation: the central problem is that of giving methods for proving inequations. This problem is nontrivial, since " $\neq$ " is not just the syntactic nonidentity of terms, nor is it an independent relation; inequations interact with equations, producing new (inequational) consequences. We address the following crucial question: how can we infer any inequation which is the logical consequence of a given system of equations and inequations?

Most current equational languages circumvent the issue of inequations by treating the " $\neq$ " relation as the logical negation of an explicitly defined equality predicate for each domain. This method is not generally applicable, and leads to unnatural specifications. For instance, $p \neq q$ is then expressed as "$eq(p, q) =$ **false**" [27]. Such an approach makes an artificial distinction between the "eq" predicate for the domain and the logical equality relation on terms defined by the specification. That is awkward because there is usually no inherent reason to define the equality predicate separately; the notion of equality is directly defined by the specification, whose axioms are viewed as formulas in equational first order logic. Current approaches to studying inequations broadly fall into the following categories:

(1) refutational techniques which derive a contradiction after adding the skolemized negation of the goal to the system [14, 30];

(2) the default assumption of two terms being not equal when their equality cannot be proved using a reasonable inference system [24, 26];

(3) finding a ground substitution which "solves" inequations in the empty theory (i.e., given $s \neq t$, finding σ such that $s\sigma, t\sigma$ are distinct ground terms) [5, 6, 19].

1.3. Overview and Main Results

In this paper, we investigate a new *forward* approach for deriving inequational consequences, similar to the one (proposed by Birkhoff [2]) traditionally used for deriving equational consequences. Our approach is different from (2) in that we derive inequations which hold in *every* model of a given system. Unlike (3) and (2), we study arbitrary finitely specified equational theories. Unlike (1), the refutational approach, inequations are deduced directly from given axioms using inference

rules. We refer to such proofs of inequations as *forward* proofs as opposed to refutational proofs. We formulate inference rules that are *forward complete*, i.e., every logical consequence of a set of equations and inequations is deducible by repeatedly applying these inference rules.

Forward proofs of equational consequences can be constructed using the traditional equality inference rules: reflexivity, symmetry, transitivity, and substitutivity. A natural way to construct forward proofs of inequations is to use inference rules (I_k) that are formulated as contrapositives of the equational inference rules (E_k), given by the following schema (where A denotes a set of equations):

$$(E_k)\ \frac{a = b,\ A}{c = d} \quad \rightarrow\rightarrow \quad (I_k)\ \frac{c \neq d,\ A}{a \neq b}$$

For example, from the substitutivity inference rule of equality, one can construct an analogous inference rule for inequations as follows:

$$\frac{s_1 = t_1,\ s_2 = t_2}{f(s_1, s_2) = f(t_1, t_2)} \quad \rightarrow\rightarrow \quad \frac{s_1 = t_1,\ f(s_1, s_2) \neq f(t_1, t_2)}{s_2 \neq t_2}$$

There are several reasons for preferring forward proofs of inequations to refutational proofs. Unlike refutational proofs, forward proofs do not *perturb* the program/specification environment by adding a skolemized literal. Such perturbation complicates matters when the property being proved is just a subgoal in a larger proof, e.g., when it is necessary to prove the antecedent of a conditional equation which contains inequations. If refutational methods are used, all other consequences of adding the new (skolemized) literal must be discarded before continuing the proof. On the contrary, all results obtained using forward proofs are logical consequences and can be stored and used subsequently in proofs of other inequations and equations. Forward proofs are conceptually simpler, and have a more appealing constructive flavor which is missing in refutational proofs which survive on excluded middles. In this aspect, they resemble proofs in natural deduction systems [10, 20]. We believe that the insight provided by the study of forward proof methods for inequations will be helpful in developing better proof procedures for them. The procedure developed in this paper is based on a combination of the forward proof method studied here and an important result we prove about the structure of proofs of inequations.

The main results of the paper are the following:

(1) A refutationally complete basic set of inference rules R_0 is obtained in Section 3 by taking the contrapositives of a set of traditionally used equational inference rules. It is shown, however, that R_0 is not forward complete: not all inequational consequences are deducible using R_0, because the contrapositive of the substitutivity inference rule is not sufficiently powerful. For example, although we can deduce $f(s, t) = f(t, s)$ from $s = t$ using the substitutivity inference rule (E4), we cannot deduce $s \neq t$ from $f(s, t) \neq f(t, s)$ using its contrapositive.

(2) In Section 4, we develop a forward complete set of inference rules R_ω by generalizing the subterm inference rule for inequations. R_ω introduces

inference rules for deducing a *conditional consequence* relation " $\underset{l,r}{\sim}$ "; intuitively, $t_1 \underset{l,r}{\sim} t_2$ signifies that $t_1 = t_2$ holds in the theory if $l = r$ is assumed. In R_ω-proofs, an inequation $l \neq r$ is deduced from $m \neq n$ and $m \underset{l,r}{\sim} n$. The proofs in R_ω are like natural deduction proofs in that they use a special set of inference rules for deducing *conditional equations*, i.e., equations with the assumption $l = r$.

(3) In Section 5, we formulate a semidecision procedure *ProveInequation* for deriving inequational consequences of a given system, using the result that every inequational consequence has an R_ω-proof in which exactly one inequational axiom is used. From this result, it immediately follows that to check for the consistency of a system of equations and inequations, it is sufficient to check, for each inequation i, whether the combination of all the equations with the inequation i is consistent. This is an instance of a phenomenon that Lassez and McAloon have called "independence of negative constraints" in [22], which appears to occur in a variety of domains with a variety of constraints. In [5] this is shown for solving a system of equations and inequations in the empty theory; in [22] it is shown for linear constraints over the reals. Such a result can often help in improving the efficiency of decision procedures. In particular, consistency of a system of equations and inequations can be checked in parallel, as inequations do not interact with each other in the consistency check.

(4) If Ξ is the subset of equations in the given system S, and if the equational theory T of $\Xi \cup \{skolem(s = t)\}$ has a (decidable) T-unification algorithm, then we have a decision procedure for determining whether $S \models s \neq t$.

(5) In Section 5, we also formulate a semidecision procedure *SolveInequation* for solving inequations in the theory defined by a given system.

2. TECHNICAL PRELIMINARIES

In this section, we present the definitions, notation, and equational inference rules used in the rest of the paper. We also state the forward completeness result for equational inference rules, and formalize a few useful properties regarding the structure of proofs of equations and inequations.

2.1. Terms, Substitutions, and Unifiers

The language considered contains a finite number of *function symbols* ($\in F$), disjoint from a (denumerably infinite) set of *variables* ($\in V$). We denote variables by characters u, v, w, x, y, z toward the end of the alphabet, possibly subscripted. A *term* is a variable, or is a constant ($\in F$), or is of the form $f(t_1, \ldots, t_n)$ where $f \in F$ and $t_1, \ldots, t_n$ are (argument) terms. By $s \equiv t$, we denote that the terms s and t are identical; contrariwise, $s \not\equiv t$ denotes that they are not identical. A term s is a *subterm* of another term t iff either $s \equiv t$, or s is the subterm of an argument of t. A term s is a *proper subterm* of t iff s is a subterm of t but $s \not\equiv t$. We write $t[s]$ to

indicate that s is a subterm of t; replacement in t of an occurrence of the subterm s by p is denoted $t[s \leftarrow p]$.

A *substitution* is a mapping from variables to terms, denoted by the symbols $\pi, \rho, \sigma, \theta$ (possibly subscripted). The *identity* substitution maps every variable to itself. The result of *applying* a substitution σ to a term t is another term $(t\sigma)$ obtained by simultaneously replacing all occurrences of the variables in t by the terms to which σ maps them. *Skolemization* of a term t is the application of a special substitution *skolem* mapping all variables of t to new constants ($\notin F$). The *composition* of substitutions is denoted by concatenating them; if $t_1\sigma_1 \equiv t_2$ and $t_2\sigma_2 \equiv t_3$, then $t_1(\sigma_1\sigma_2) \equiv t_3$. A substitution σ is *more general* than the substitution ρ iff $\exists\theta\forall t[t\sigma\theta \equiv t\rho]$. Among a given class of substitutions Σ, a substitution $\sigma \in \Sigma$ is *most general* iff $(\forall\rho \in \Sigma)\exists\theta\forall t[t\sigma\theta \equiv t\rho]$. Two terms p, q are *unifiable* iff they have a *unifier*, i.e., a substitution σ such that $p\sigma \equiv q\sigma$. Every pair of unifiable terms has a *most general unifier* (m.g.u.) which is unique up to renaming variables.

2.2. Equations, Inequations, and Interpretations

An *equation* (or positive literal) is a two-tuple of terms written with the infix operator " = ". Similarly, an *inequation* (or negative literal) is a two-tuple of terms separated by " ≠ ". A *literal* is either an equation or an inequation. A *system* is a finite set of literals. An *equational* system contains only equations, while an *inequational* system contains only inequations.

An *interpretation* μ is a mapping from ground literals to {**true**, **false**}, and is said to be *consistent* if $\mu(p = q) \not\equiv \mu(p \neq q)$ for all ground terms p, q. A literal P *holds* in μ if $\mu(\mathrm{P}\sigma) =$ **true** for every ground substitution σ. An *E-interpretation* of a system S is a consistent interpretation μ such that every literal in the reflexive, symmetric, transitive, substitutive closure of the equality relation defined by S holds in μ. We say $S \models \mathrm{P}$ ("S *entails* P") if P holds in every E-interpretation of S. S is *E-unsatisfiable* (or inconsistent) if it has no E-interpretation; otherwise, S is *E-satisfiable*.

The *T-unification* of a given pair of (possibly *T-unifiable*) terms s, t with respect to a set T of equational FOPC formulas is the problem of finding a *T-unifier*, i.e., a substitution σ such that $T \models (s\sigma = t\sigma)$. Note that T-unifiable terms may not have a unique most general T-unifier.

2.3. Proof Trees

A *forward proof* of a literal P from a system S, using a set of inference rules R, is a proof tree whose root is that literal P, and whose nonempty leaves are axioms ($\in S$), such that every nonleaf node is a literal obtained by applying one of the inference rules in R to the literals that are the children of that node. When such a forward proof of P exists, we say $S \vdash_R \mathrm{P}$ ("S derives P using R"), omitting the subscript R when the context is clear. Proof trees are depicted with child nodes just above their parent node. R is *forward complete* ("complete", for short) if $S \models \mathrm{P} \supset S \vdash_R \mathrm{P}$ for every system S and every literal P.

A *refutation(al) proof* of a literal P from a system S using a set R of inference rules is the forward proof of $\square$ (or some instance of $x \neq x$) from the system

$S \cup \{skolem(\neg P)\}$ using R. The set R is said to be *refutationally complete* if for every literal P and system S, $S \models P$ implies that $S \cup \{skolem(\neg P)\} \vdash_R \square$.

2.4. Equational Inference Rules

The following is a complete set of simple equational inference rules which can be used for deducing equations from other equations.

Theorem 1. Every equational consequence of a system can be deduced using $\mathbf{E} = \{E0, E1, E2, E3, E4\}$, *consisting of the following inference rules*:

Reflexivity: (E0) $\dfrac{}{p = p}$

Symmetry: (E1) $\dfrac{p = q}{q = p}$

Transitivity: (E2) $\dfrac{p = q,\ q = r}{p = r}$

Instance: (E3) $\dfrac{p = q}{p\sigma = q\sigma}$

Superterm: (E4) $\dfrac{p_1 = q_1, \ldots,\ p_n = q_n}{f(p_1, \ldots, p_n) = f(q_1, \ldots, q_n)}$

PROOF. See [2]. □

2.5. Structure of Proofs

Here we characterize a few useful properties of the general structure of the proof trees in our inference systems. These will be used later in the proofs of several of our theorems. Proposition 1 states that every proof of an equation using the inference rules in **E** can be organized so that reflexivity (E0), symmetry (E1), and instance (E3) steps can all be done before any of the superterm (E4) inference steps are performed, and all the superterm (E4) steps are done prior to any of the transitivity inferences (E2).

Proposition 1. The proof tree of any equation can be transformed into another proof tree for the same equation in which applications of transitivity (E2) *are not followed by those of any other inference rule, and applications of the superterm rule* (E4) *are followed only by transitivity* (E2).

PROOF. The proof consists of performing a series of transformations on a proof tree to move all the (E2) steps towards the root, followed by another series of transformations to push the (E4) steps. The details are given in Section A.3 of the Appendix. □

Inequational inference rules typically use one inequation and zero or more equations to deduce another inequation. Such inference rules are of no use in deriving equations; conversely, inequations are neither premises nor results in

applications of equational inference rules ($\in \mathbf{E}$). The following proposition formalizes these useful observations.

Proposition 2. Let R be a set of inference rules consisting of the rules in $\mathbf{E} = \{E0, E1, E2, E3, E4\}$ *as well as any other inference rules (for deducing inequations) with the following structure:*

$$\frac{l \neq r,\ s_1 = t_1,\ s_2 = t_2, \ldots,\ s_k = t_k}{p \neq q}.$$

Then, for any system A and any terms p, q, we have the following:

(2.1) An independence result for deduction of equations: IF $A \vdash_R p = q$ THEN *there is a proof of* $p = q$ *from A using R which uses no inequational axiom* $\in A$ *and involves only the inference rules in* **E**.

(2.2) An independence result for deduction of inequations: IF $A \vdash_R p \neq q$ THEN *there is a proof of* $p \neq q$ *from A using R which uses some equations* $\in A$ *and exactly one inequation* $\in A$.

PROOF. (2.1): In each of the equational inference rules (E0–E4), no premise is an inequation. In other inference rules of the kind given above, the inferred literal is not an equation. Hence every literal in the proof tree of an equation is an equation.

(2.2): There is exactly one inequation among the premises of each inference rule which infers an inequation. Hence, in the proof tree of an inequation, at most one literal at each level is an inequation, and precisely one leaf of the proof tree must be an inequation. The proof tree has a unique chain of inequations, from the root to a leaf. □

A stronger version of (2.2) is proved later as Theorem 3.

As a special case, substitute p for q in (2.2) above: if $A \vdash_R p \neq p$, then there is a proof of $p \neq p$ which invokes zero or more equations and exactly one inequation in A. So if there are no inequational axioms, $p \neq p$ cannot be deduced and the system is consistent. Since $p = p$ is obtained by equational reflexivity, any derivation of $p \neq p$ (abbreviated "□") using sound inference rules indicates that the system is inconsistent.

3. INFERENCE RULES FOR INEQUATIONS

We first introduce a set of inference rules for deducing inequations which are inspired by the equational inference rules ($\in \mathbf{E}$) given earlier. We show (Theorem 4) the refutational completeness of a subset (*basic set* R_0) of these inference rules along with **E**. We also prove a result (Theorem 3) which is used in proving the main results that follow in subsequent sections. This theorem shows that a refutation proof can always be organized into a certain standard form in which only one of the inequation axioms of the system is used.

The following are inference rules that derive inequations, with an inequation as one of the premises. I1, I2, $I4^I$ are the contrapositives of the equational inference rules E1, E2, E4 respectively. I3, like E3, allows instantiation, implementing im-

plicit universal quantification of variables. I0 is a redundant rule, indicating that "$\square$" represents a contradiction.

Refutation: (I0) $\dfrac{p \neq p}{\square}$.

Symmetry: (I1) $\dfrac{p \neq q}{q \neq p}$.

Transitivity: (I2) $\dfrac{p \neq q,\ q = r}{p \neq r}$.

Instance: (I3) $\dfrac{p \neq q}{p\sigma \neq q\sigma}$.

Subterm: (I4[I]) $\dfrac{f(p_1, \ldots, p_k, \ldots, p_n) \neq f(q_1, \ldots, q_k, \ldots, q_n),\ p_1 = q_1, \ldots, p_{k-1} = q_{k-1}, p_{k+1} = q_{k+1}, \ldots, p_n = q_n}{p_n \neq q_k}$.

Theorem 2 (Soundness of inequational inference rules). Every inequational inference rule given above is sound: if the premises of a rule hold in all E-interpretations of a system, then so does the consequence.

PROOF. Inference rule I0 is sound because $(\forall p)\ p = p$ holds in every E-interpretation; there is no consistent interpretation in which $p \neq p$ holds.

(1) Let $A \models p \neq q$. In no E-interpretation of A is it possible that $p\sigma = q\sigma$ for any σ. The symmetry of equality implies that $q\sigma = p\sigma$ also cannot hold for any σ in any E-interpretation of A. Hence $A \models q \neq p$, and I1 is a sound inference rule.

(2) Similarly let $A \models p \neq q$ as well as $A \models q = r$. This implies that $p\sigma \neq q\sigma$ and $q\sigma = r\sigma$ hold in every E-interpretation for A, for every substitution σ. It is impossible that $p\sigma = r\sigma$ holds in any such E-interpretation, since equational symmetry and transitivity would then imply that $p\sigma = q\sigma$ holds, contradicting the consistency of the E-interpretation in which $p\sigma \neq q\sigma$ holds. Thus, instances of p and r are not equal in any E-interpretation for A, so that $A \models p \neq r$, and inference rule I2 is sound.

(3) Soundness of I3 is straightforward, since all variables in $p \neq q$ are assumed universally quantified: if $A \models (\forall x.p \neq q)$, then $A \models p\sigma \neq q\sigma$ for any substitution σ.

(4) Let $A \models f(p_1, \ldots, p_n) \neq f(q_1, \ldots, q_n)$ and also $A \models p_i = q_i\ \forall i \neq k\ (1 \leq i \leq n)$, implying that every instance of $p_i = q_i$ holds in every E-interpretation of A. Suppose A has some E-interpretation μ in which $p_k\sigma = q_k\sigma$ holds, for some substitution σ. Then, $f(p_1, \ldots, p_n)\sigma = f(q_1, \ldots, q_n)\sigma$ holds in μ, since E-interpretations contain the substitutive closure of equations. This contradicts $A \models f(p_1, \ldots, p_n) \neq f(q_1, \ldots, q_n)$, the initial assumption. Hence, instances of p_k, q_k cannot be equal to any E-interpretation of A. Hence $A \models p_k \neq q_k$, and I4[I] is sound. $\square$

The study of *completeness* of the inequational inference rules is the main concern of the rest of this section and the next section (which concentrates on forward completeness). We start by considering the *basic set* of inference rules $R_0 = \mathbf{E} \cup \{I0, I1, I2, I3\}$. In the next section we will consider the set $R_1 = R_0 \cup \{I4^1\}$, which includes the subterm inference rule, as a starting point towards forward completeness.

Theorem 3, below, shows that from every inconsistent system S one can construct a "canonical" refutational proof using the rules in R_0 such that

(1) exactly one inequation from S is used in the proof, and

(2) the only inequational inference rules used are the instantiation rule (I3) followed by a single step of the transitivity rule (I2).

The existence of such a canonical R_0-proof tree is crucial for showing the forward completeness theorem in the next section, and for the semidecision procedure for inequations to be developed later. Note that this result strengthens the independence result for deduction of inequations stated in Proposition 2.2 earlier. Theorem 4 shows the refutational completeness of R_0.

Theorem 3. If $A \vdash_{R_0} \square$, *then* $A \vdash_{R_0} (p\rho = q\rho)$ *for some inequation* $(p \neq q) \in A$ *and some substitution* ρ.

PROOF. We show that whenever $A \vdash_{R_0} \square$, any R_0-proof of $\square$ from A can be transformed to the following form, which includes as a subtree the R_0-proof of $q\rho = p\rho$ for some substitution ρ, where $p \neq q \in A$. By symmetry (E1), we hence have $A \vdash_{R_0} p\rho = q\rho$.

$$\text{(I2)}\ \frac{\begin{array}{c}\vdots\quad\vdots \\ q\rho = p\rho\end{array} \qquad \text{(I3)}\ \frac{p \neq q}{p\rho \neq q\rho}}{q\rho \neq q\rho}$$

Note that from Proposition 2.2 (the independence result for inequations) the R_0-derivation of $\square$ invokes a unique inequation $p \neq q \in A$. Starting from such an R_0-proof of $\square$, we apply a series of transformations on the tree to "pull down" the inequation $p \neq q$ towards the root until it is separated from the root by no more than one application of rule I3 (instantiation). The resulting proof tree has the desired structure, with $q\rho = p\rho$ and $p\rho \neq q\rho$ at the child nodes of the contradiction.

The first set of transformations applied transforms the tree so that all instantiation (I3) steps are done right in the beginning, followed by symmetry (I1) before any of the transitivity (I2) steps. Thus, all instantiations of inequations can be combined into a single substitution (ρ). The fact that such a transformation can always be done without changing the outcome of the proof tree is shown in Lemma 3.3 given in the Appendix.

Lemma 3.4 (given in Section A.2 of the Appendix) shows that in a *transitive* proof tree, i.e., an R_0-proof tree that uses only the transitive (E2, I2) and symmetry (E1, I1) steps, one can always move a given leaf until it is no more than one hop from the root. We apply the transformations of the kind described in Lemma 3.4 to the transitive part of the proof tree obtained previously, to reposition $p\rho \neq q\rho$ so that it is one hop from the root, thereby getting the proof tree into the desired form. $\square$

Theorem 4 (Refutational completeness of R_0). If $S \models s \neq t$, then

$$S \cup \{\text{skolem}(s = t)\} \vdash_{R_0} \square .$$

PROOF. The proof is based on the well-known result [4] that for an E-unsatisfiable set of clauses S, there is a proof of $\square$ from $S \cup \{x = x\} \cup F$, using the inference rules of positive hyperresolution and hyperparamodulation, where F is the set of functionally reflexive axioms for S. We show that every such resolution proof of refutation can be transformed into an equivalent proof which uses the inference rules in R_0. Details are given in Section A.1 of the Appendix. $\square$

4. A FORWARD COMPLETE INFERENCE SET

In this section, we develop a forward complete set of inference rules (R_ω). The basic set R_0 is clearly not complete, since it does not contain any rule contrapositive to the superterm rule (E4). However, $R_1 = R_0 \cup \{\text{I4}^1\}$ is also incomplete; so are sets of inference rules which include generalized versions of I4^1. After progressive generalization of the subterm rule, we have forward completeness in the limit for R_ω which contains inference rules capable of deducing "conditional consequences".

Proposition 3. R_1 is not forward complete.

PROOF. By the example shown below. $\square$

Example 4.1. $S_0 = \{f(a, b) \neq f(b, a)\}$; prove $a \neq b$.

Inference rules in R_1 cannot be applied to infer any useful new inequation from S_0. Hence $a \neq b$ is not derivable, although $\{f(a, b) \neq f(b, a)\} \models (a \neq b)$.

4.1. A First Attempt

Definition. Two terms m, n are *l, r-identical* (denoted $m \underset{l,r}{\approx} n$) if m and n can be made syntactically identical by zero or more mutual replacements in them of occurrences of l by r and of r by l respectively.

For example, we have

$$f(b, x, g(a, b)) \underset{a,b}{\approx} f(a, x, g(b, a)) \quad \text{and} \quad h(h(h(a))) \underset{h(a),a}{\approx} h(a).$$

The significance of the $\underset{l,r}{\approx}$ relation is that two terms are l, r-identical iff their equality can be deduced from $l = r$ using the rules of equality (E) excluding instantiation. A similar "constant congruence" relation was used in [3] and in [31]. The indices l, r of $\underset{l,r}{\approx}$ are sometimes omitted when the context is clear.

Lemma 1 (The $\underset{l,r}{\approx}$ lemma). Let l, r be ground terms. Then

$$\{l = r\} \vdash_{R_0} \text{m} = \text{n} \quad \textit{iff} \quad \text{m} \underset{l,r}{\approx} \text{n}.$$

PROOF. $\Rightarrow$: Suppose $\{l = r\} \vdash_{R_0} m = n$, where l, r are ground. Only the equational inference rules (E0)–(E4) can be used to derive an equation, i.e., $l = r \vdash_E m = n$, by Proposition 2.1. Each of these equational inference rule preserves the property that if the terms in the premises are l, r-identical, then so are the inferred literals:

(E0): Trivially, $s \underset{l,r}{\approx} s$.

(E1): If $s \underset{l,r}{\approx} t$, then $t \underset{l,r}{\approx} s$.

(E2): If $s \underset{l,r}{\approx} t$ and $t \underset{l,r}{\approx} p$, then $s \underset{l,r}{\approx} p$.

(E3): If $s \underset{l,r}{\approx} t$, then $s\sigma \underset{l,r}{\approx} t\sigma$ (since l and r are ground).

(E4): If $\forall i[s_i \underset{l,r}{\approx} t_i]$, then $f(s_1, \ldots, s_n) \underset{l,r}{\approx} f(t_1, \ldots, t_n)$.

$\Leftarrow$: The converse holds (by definition of $\underset{l,r}{\approx}$) even if l, r are nonground, because replacements of equals by equals (l by r, and r by l, in this case) are sound equational inferences, given the system $\{l = r\}$. □

The restriction that l and r should be ground is needed for the first part ($\Rightarrow$) of the above proof. This is because the check for $\underset{l,r}{\approx}$ does not allow instantiation, hence we cannot conclude that instances of l and r are l, r-identical. For example, $f(a)$ and $g(a)$ are *not* $f(x), g(x)$-identical, although $\{f(x) = g(x)\} \vdash_E f(a) = g(a)$. We now use the $\underset{l,r}{\approx}$ relation to devise generalizations of the subterm inference rule to help formulate a forward complete set of inference rules.

Let R_2 be $R_0 \cup \{I4^{II}\}$, with the new inference rule

$$(I4^{II}) \quad \frac{m \neq n,\ m \underset{l,r}{\approx} n}{l \neq r}.$$

With the subterm rule generalized to $I4^{II}$, we can prove (using R_2) the inequation unprovable by R_1 in Example 4.1. The next theorem states the forward completeness of R_2 for inequational systems, i.e., systems that consist only of inequations. Although such systems are restricted, they are useful in several situations. For example, the following system characterizes the fact that the sorts **integer** and **list** must be distinct: $\{0 \neq nil,\ 0 \neq cons(x, y),\ succ(z) \neq nil,\ succ(z) \neq cons(x, y)\}$.

Theorem 5. R_2 is (forward) complete for inequational systems, i.e., if Q *is any set of inequations, then* $Q \models (s \neq t) \Rightarrow Q \vdash_{R_2} (s \neq t)$.

PROOF. Let $Q \models (s \neq t)$, so that refutational completeness (Theorem 4) implies $Q \cup \{s\sigma = t\sigma\} \vdash_{R_0} \square$, where σ is a skolem substitution for s, t. By Theorem 3, there is a substitution ρ such that $Q \cup \{s\sigma = t\sigma\} \vdash_{R_0} [p\rho = q\rho]$ such that $(p \neq q) \in Q$. By Proposition 2.1 (the independence result for equational deductions), we then have $\{s\sigma = t\sigma\} \vdash_{R_0} p\rho = q\rho$. We conclude (from Lemma 1) that $p\rho \underset{s\sigma, t\sigma}{\approx} q\rho$. Using inference rule $I4^{II}$, we infer $s\sigma \neq t\sigma$. Since σ is a skolemizing substitution, this is equivalent to a deduction of $s \neq t$. □

Although several more inferences are possible now than before, R_2 is still not forward complete for systems containing equations as well as inequations, as the following example shows.

Example 4.2. $S_1 = \{g(a) = h(a),\ g(b) \neq h(b)\}$; prove $a \neq b$.

Here, although $S_1 \cup \{a = b\}$ derives a contradiction, implying that $S_1 \models (a \neq b)$, no other useful inequation is derivable from S_1 even if we use $\mathrm{I4}^{\mathrm{II}}$. To account for this example, we may use $R_3 = R_0 \cup \{\mathrm{I4}^{\mathrm{III}}\}$, generalizing $\mathrm{I4}^{\mathrm{II}}$ to the stronger inference rule $\mathrm{I4}^{\mathrm{III}}$ defined below:

$$(\mathrm{I4}^{\mathrm{III}})\quad \frac{m \neq n,\ m \underset{l,r}{\approx} m_1,\ n \underset{l,r}{\approx} n_1,\ m_1 = n_1}{l \neq r}.$$

For Example 4.2 above, we then have the proof

$$(\mathrm{I4}^{\mathrm{III}})\quad \frac{g(b) \neq h(b),\ g(b) \underset{a,b}{\approx} g(a),\ h(b) \underset{a,b}{\approx} h(a),\ g(a) = h(a)}{a \neq b}.$$

But even R_3 is incomplete, and sometimes does not succeed in deducing inequational consequences from a system. The following example shows that there are inequations for which there is a refutational proof but no forward proof using the inference rules in R_3.

Example 4.3. $S_2 = \{f(a) = h(a),\ f(b) = g(b),\ g(a) \neq h(b)\}$; prove $a \neq b$.

Clearly, $S_2 \cup \{a = b\} \vdash_{R_0} \square$, hence $S_2 \models (a \neq b)$. But no other inequation can be inferred from S_2 using the inference rules of R_3, and nor can any of the equations needed for applying $\mathrm{I4}^{\mathrm{III}}$. So $R_0 \cup \{\mathrm{I4}^{\mathrm{III}}\}$ is not sufficiently powerful to derive $a \neq b$, and can be further strengthened; we can infer $a \neq b$ using the inference rule $\mathrm{I4}^{\mathrm{IV}}$ given below:

$$(\mathrm{I4}^{\mathrm{IV}})\quad \frac{m \neq n,\ m \underset{l,r}{\approx} m_1,\ n \underset{l,r}{\approx} n_1,\ m_1 = m_2,\ n_1 = n_2,\ m_2 \underset{l,r}{\approx} n_2}{l \neq r}.$$

However, even $R_0 \cup \{\mathrm{I4}^{\mathrm{IV}}\}$ is not forward complete, and fails to deduce the logical consequence $a \neq b$ from

$$S_3 = \{f(a) = h(a),\ f(b) = k(a),\ k(b) = g(b),\ g(a) \neq h(b)\}.$$

We can thus define $\mathrm{I4}^{\mathrm{III}}, \mathrm{I4}^{\mathrm{IV}}, \mathrm{I4}^{\mathrm{V}}, \ldots$, ad infinitum, so that for any $\mathrm{I4}^{N}$, we have a stronger inference rule $\mathrm{I4}^{N+1}$ which helps prove more inequations. Each inference rule is obtained from the previous one by incorporating one more instance of transitivity between $=$ and $\approx$. While $m_{k-1} = n_{k-1}$ is one of the premises of $\mathrm{I4}^{2k-1}$, we have $m_{k-1} = m_k \approx n_k = n_{k-1}$ in the premises of $\mathrm{I4}^{2k}$; similarly, while $m_k \approx n_k$ is a premise of $\mathrm{I4}^{2k}$, we have $m_k \approx m_{k+1} = n_{k+1} \approx n_k$ in the premises of $\mathrm{I4}^{2k+1}$. The following progression illustrates this, showing the chain of reasoning

which deduces $l \neq r$ using each of $I4^{II}$–$I4^{V}$:

$I4^{II}$

$$m \underset{\approx}{\neq} n$$

$I4^{III}$

$$\begin{array}{cc} m & \neq & n \\ \approx & & \approx \\ m_1 & = & n_1 \end{array}$$

$I4^{IV}$

$$\begin{array}{cc} m & \neq & n \\ \approx & & \approx \\ m_1 & & n_1 \\ \| & & \| \\ m_2 & \approx & n_2 \end{array}$$

$I4^{V}$

$$\begin{array}{cc} m & \neq & n \\ \approx & & \approx \\ m_1 & & n_1 \\ \| & & \| \\ m_2 & & n_2 \\ \approx & & \approx \\ m_3 & = & n_3 \end{array}$$

4.2. *Conditional Consequence Mechanism*

Each of the sets of inference rules $R_k \equiv R_0 \cup \{I4^k\}$ discussed above can be shown to be incomplete. One apparent problem is that we do not have transitivity between $\approx$ and $=$ relations. To handle a proof with any number of occurrences of $\approx$-literals, we need a rule (or a set of rules) that encapsulates every rule in the series $\{I4^k\}$, $k > I$.

For this purpose we introduce a new relation called *l, r-equivalence* ($\underset{l,r}{\sim}$), deduced using the following inference rules. We show that $\underset{l,r}{\sim}$ is a conditional equality, i.e., $S \cup \{l = r\} \vdash m = n \supset S \vdash m \underset{l,r}{\sim} n$. We then prove the forward completeness of the new inference system R_ω obtained by adding these inference rules to R_0.

Definition. $R_\omega(l, r) = R_0 \cup \{E0^\omega_{l,r}, E2^\omega_{l,r}, E4^\omega_{l,r}, E5^\omega_{l,r}, I4^\omega_{l,r}\}$, where the new inference rules are as defined below:

$$(E0^\omega_{l,r}) \quad \frac{m \underset{l,r}{\approx} n}{m \underset{l,r}{\sim} n}$$

$$(I4^\omega_{l,r}) \quad \frac{m \underset{l,r}{\sim} n,\ m \neq n}{l \neq r}$$

$$(E5^\omega_{l,r}) \quad \frac{m = n}{m \underset{l,r}{\sim} n}$$

$$(E2^\omega_{l,r}) \quad \frac{m \underset{l,r}{\sim} n,\ n \underset{l,r}{\sim} p}{m \underset{l,r}{\sim} p}$$

$$(E4^\omega_{l,r}) \quad \frac{p_1 \underset{l,r}{\sim} q_1, \ldots, p_n \underset{l,r}{\sim} q_n}{f(p_1, \ldots, p_n) \underset{l,r}{\sim} f(q_1, \ldots, q_n)}$$

Example 4.4 (Illustration of R_ω). Let S contain the following literals:

(a) $0 + x = x$,

(b) $s(x) + y = s(x + y)$,

(c) $x + y = y + x$,

(d) $x + (y + z) = (x + y) + z$,

(e) $x + (x + x) = 0$,

(f) $0 \neq s(0)$.

Equations (a),(b) define "+"; (c),(d) state that "+" is associative and commutative; (e) states that $x + x$ is the additive inverse of x; and the last equation (f) specifies that 0 and s(0) are two distinct objects.

TASK. To obtain the forward proof of $s(0) \neq s(s(0))$ using the inference rules of $R_\omega(s(0), s(s(0)))$, which should be possible if R_ω is complete, since $S \models s(0) \neq s(s(0))$. The following R_ω-proof comprises three parts (some parentheses and subscripts are omitted):

(1) By (b), we have $ss0 + (s0 + s0) = s(s0 + (s0 + s0))$. The latter term equals s(0), because $s0 + (s0 + s0)$ is an instance of $x + (x + x)$, which equals 0, by equation (e) in S. Hence $s0 \underset{s0, ss0}{\sim} ss0 + (s0 + s0)$ by $E5^{\omega}_{s0, ss0}$.

(2) $ss0 + (s0 + s0)$ is s0, ss0-identical to $s0 + (s0 + s0)$, which equals 0, applying equation (e). Hence $ss0 + (s0 + s0) \underset{s0, ss0}{\sim} 0$ by $E0^{\omega}_{s0, ss0}$.

(3) $s0 \underset{s0, ss0}{\sim} 0$ follows from these two s0, ss0-equivalences. Since $s0 \neq 0$ is an axiom in S, we finally derive $s0 \neq ss0$ by $I4^{\omega}_{s0, ss0}$.

The full proof trees are shown below:

$$(E2)\ \dfrac{(E3)\ \dfrac{(E4)\ \dfrac{(E1)\ \dfrac{x + (x + x) = 0}{0 = x + (x + x)}}{s0 = s(x + (x + x))}}{s0 = s(s0 + (s0 + s0)),}\quad (E3)\ \dfrac{(E1)\ \dfrac{sx + y = s(x + y)}{s(x + y) = sx + y}}{s(s0 + (s0 + s0)) = ss0 + (s0 + s0)}}{(E5^{\omega}_{s0, ss0})\ \dfrac{s0 = ss0 + (s0 + s0)}{s0 \underset{s0, ss0}{\sim} ss0 + (s0 + s0)}},$$

$$(E2^{\omega})\ \dfrac{(E0^{\omega})\ \dfrac{ss0 + (s0 + s0) \underset{s0, ss0}{\approx} s0 + (s0 + s0)}{ss0 + (s0 + s0) \underset{s0, ss0}{\sim} s0 + (s0 + s0),}\quad (E5^{\omega})\ \dfrac{(E3)\ \dfrac{x + (x + x) = 0}{s0 + (s0 + s0) = 0}}{s0 + (s0 + s0) \underset{s0, ss0}{\sim} 0}}{ss0 + (s0 + s0) \underset{s0, ss0}{\sim} 0},$$

$$(I4^{\omega}_{s0, ss0})\ \dfrac{(E2^{\omega})\ \dfrac{s0 \underset{s0, ss0}{\sim} ss0 + (s0 + s0),\ ss0 + (s0 + s0) \underset{s0, ss0}{\sim} 0}{s0 \underset{s0, ss0}{\sim} 0,}\quad (I1)\ \dfrac{0 \neq s0}{s0 \neq 0}}{s0 \neq ss0}.$$

4.3. Soundness and Completeness of R_ω

4.3.1. Proof of Soundness. The *soundness* of the inference rules $E0^{\omega}_{l,r}$, $E2^{\omega}_{l,r}, E4^{\omega}_{l,r}, E5^{\omega}_{l,r}$ follows from the soundness of the equational inference rules ($\in \mathbf{E}$) and the "only if" part ($\Leftarrow$) of Theorem 6 which follows. The rule $I4^{\omega}_{l,r}$ is sound because

$$\textit{if} \quad S \vdash_{R_\omega} \mathrm{m} \underset{l,r}{\sim} \mathrm{n} \quad \textit{then} \quad S \cup \{l = r\} \vdash_{R_0} \mathrm{m} = \mathrm{n} \qquad \text{(by Corollary 6.2 below)},$$

implying $S \cup \{l = r, \mathrm{m} \neq \mathrm{n}\} \vdash_{R_0} \square$, hence $S \cup \{\mathrm{m} \neq \mathrm{n}\} \models l \neq r$.

Theorem 6. Let l, r be ground terms. Then

$$\left[S \cup \{l = r\} \vdash_{R_0} \mathrm{m} = \mathrm{n}\right] \quad \textit{iff} \quad \left[S \vdash_{R_\omega(l,r)} \mathrm{m} \underset{l,r}{\sim} \mathrm{n}\right].$$

PROOF. $\Rightarrow$: Let T be any proof tree for $\mathrm{m} = \mathrm{n}$ from $S \cup \{l = r\}$ using R_0. We first transform T into another proof tree T_0 in which applications of E0 (reflexivity), E1 (symmetry), and E3 (instance) are followed by E4 (superterm) and finally E2 (transitivity), as in Proposition 1. We observe that T_0 contains proof subtrees of equations derived from S alone and/or $\{l = r\}$ alone, which interact (become siblings in the proof tree) via applications of inference rules E2, E4. By repeatedly applying steps (1)–(3) below, T_0 is transformed into a new proof tree (T_3) which represents a proof $S \vdash_{R_\omega(l,r)} \mathrm{m} \underset{l,r}{\sim} \mathrm{n}$.

(1) In the lower part of T_0, consisting of equations obtained after applying E4 and/or E2, change $p = q$ to $p \underset{l,r}{\sim} q$ in every node. This corresponds to changing applications of E2 and E4 to $E2^{\omega}_{l,r}$ and $E4^{\omega}_{l,r}$, respectively. Let the result be the tree T_1.

(2) In T_1, whenever $p = q$ has a parent of the form $M \underset{l,r}{\sim} N$, insert $p \underset{l,r}{\sim} q$ between them, i.e., apply the following transformation:

$$\frac{p = q, \ldots, \ldots}{M \underset{l,r}{\sim} N} \quad \rightarrow\rightarrow \quad \frac{\dfrac{p = q}{p \underset{l,r}{\sim} q}, \ldots, \ldots}{M \underset{l,r}{\sim} N}$$

This corresponds to applying rules $E5^{\omega}_{l,r}$ and (partially) $E0^{\omega}_{l,r}$. Let tree T_2 be the result of applying this transformation to T_1.

(3) Obtain a new proof tree T_3 from T_2, replacing by

$$\frac{M \underset{l,r}{\approx} N}{M \underset{l,r}{\sim} N}$$

each subtree (with root $M = N$) in which only $l = r$ occurs at its leaves:

$$\dfrac{\dfrac{l=r}{\vdots}\ \vdots}{\dfrac{M=N}{p \underset{l,r}{\sim} q}\,,\ldots,\ldots} \quad \rightarrow\rightarrow \quad \dfrac{\dfrac{M \underset{l,r}{\approx} N}{M \underset{l,r}{\sim} N}\,,\ldots,\ldots}{p \underset{l,r}{\sim} q}$$

This transformation completes the introduction of rule $\mathrm{E0}^{\omega}_{l,r}$.

Note that l, r must be ground (as in the case of Lemma 1) for the above argument to hold. Otherwise, $l = r$ could occur as the premise in an instantiation inference rule (E3). It would then be impossible to eliminate $l = r$ from such a proof tree using any of the new R_{ω}-inference rules.

$\Leftarrow$: For each $R_{\omega}(l, r)$-proof tree, there is a corresponding R_0-proof tree in which $l = r$ is a leaf (i.e., is being used as an explicit axiom). For example, the equivalent of the deduction

$$(\mathrm{E0}^{\omega}_{l,r})\ \frac{\mathrm{m} \approx \mathrm{n}}{\mathrm{m} \sim \mathrm{n}}$$

can be obtained as a proof of $\mathrm{m} = \mathrm{n}$ by mutually replacing the corresponding occurrences of l, r in them using equational inference rules, since m, n are identical up to replacement of l, r subterms. $\mathrm{E5}^{\omega}_{l,r}$ holds trivially when "$\underset{l,r}{\sim}$" is changed to "$=$". Proof steps using rules $\mathrm{E4}^{\omega}_{l,r}, \mathrm{E2}^{\omega}_{l,r}$ have straightforward equational analogs using E4, E2 respectively, with $l = r$ being used as an axiom. □

Corollary 6.1. Let $l\sigma = r\sigma$ be a skolemized instance of an equation $l = r$. Then

$$\left[S \cup \{l\sigma = r\sigma\} \vdash_{R_0} \mathrm{m} = \mathrm{n}\right] \textit{ iff } \left[S \vdash_{R_{\omega}(l,r)} \mathrm{m} \underset{l,r}{\sim} \mathrm{n}\right].$$

PROOF. Theorem 6 guarantees the existence of an $R_{\omega}(l\sigma, r\sigma)$-proof. But an $R_{\omega}(l\sigma, r\sigma)$-proof tree of $\mathrm{m} \underset{l\sigma, r\sigma}{\sim} \mathrm{n}$ can always be trivially transformed into an $R_{\omega}(l, r)$-proof tree of $\mathrm{m} \underset{l,r}{\sim} \mathrm{n}$ by replacing the skolem constants in $l\sigma$ and $r\sigma$ by new variables distinct from every other variable used in the proof tree. □

Corollary 6.2. If $[S \vdash_{R_{\omega}(l,r)} \mathrm{m} \underset{l,r}{\sim} \mathrm{n}]$ then $[S \cup \{l = r\} \vdash_{R_0} \mathrm{m} = \mathrm{n}]$.

PROOF. Note that the proof of the $\Leftarrow$ part of Theorem 6 does not rely on l, r being ground. Hence, it holds for nonground terms also. □

4.3.2. Proof of Completeness. We need to show that $S \models P$ iff $S \vdash_{R_{\omega}} P$, where P is an arbitrary equation or inequation in the language. Since $\mathbf{E} \subseteq R_{\omega}$, it follows (by Theorem 1) that $S \models (m = n) \Rightarrow S \vdash_{R_{\omega}} (m = n)$. The other half of the proof of completeness is given as Theorem 7 below.

The following theorem states that every inequational consequence $s \neq t$ of a system S can be derived using $R_{\omega}(s, t)$. The proof uses refutational completeness of R_0, together with Theorem 3, to first show that there is an R_0-proof from

$S \cup \{skolem(s = t)\}$ of some equation $p\rho = q\rho$ that contradicts an inequation $p \neq q$ in S. By Corollary 6.1, there must be an R_ω-proof of $p\rho \underset{s,t}{\sim} q\rho$. Rule I4$^\omega$ can now be applied, yielding an R_ω-proof of $s \neq t$.

Theorem 7. IF $S \models s \neq t$, THEN $S \vdash_{R_\omega(s,t)} s \neq t$.

PROOF. Let $S \models s \neq t$. Let σ be a skolem substitution for variables in s, t. Since R_0 is refutationally complete (Theorem 4), we have

$$[S \models s \neq t] \Rightarrow (S \cup \{s\sigma = t\sigma\} \vdash_{R_0} \square).$$

By Theorem 3, there exists an inequation $p \neq q \in S$ and a substitution ρ such that

$$(S \cup \{s\sigma = t\sigma\} \vdash_{R_0} \square) \Rightarrow (S \cup \{s\sigma = t\sigma\} \vdash_{R_0} p\rho = q\rho).$$

From Corollary 6.1, we have

$$(S \cup \{s\sigma = t\sigma\} \vdash_{R_0} p\rho = q\rho) \Rightarrow (S \vdash_{R_\omega(s,t)} p\rho \underset{s,t}{\sim} q\rho).$$

Since $p \neq q \in S$ and the instantiation rule (I3) is in $R_\omega(s, t)$, we have

$$S \vdash_{R_\omega(s,t)} p\rho \neq q\rho.$$

Finally, applying $\text{I4}^\omega_{s,t}$ to $[p\rho \underset{s,t}{\sim} q\rho]$ and $[p\rho \neq q\rho]$, we have

$$S \vdash_{R_\omega(s,t)} s \neq t. \qquad \square$$

5. PROOF PROCEDURES FOR INEQUATIONS

By combining the results of Theorem 3 and Corollary 6.1 we have the following: if $s \neq t$ is a consequence of a system S, then S contains an inequation $p_i \neq q_i$ such that for some ρ we can derive $p_i\rho \underset{s,t}{\sim} q_i\rho$ using R_ω. This means that to prove $s \neq t$, we have to find a $p_i \neq q_i \in S$ and a substitution ρ such that $S \vdash_{R_\omega} p_i\rho \underset{s,t}{\sim} q_i\rho$. Here we address the issue of devising a procedure that searches for such a proof. Any such procedure must perform a substantial amount of search to find a R_ω-proof because of the two unknowns involved: (1) the substitution ρ, and (2) the inequation $p_i \neq q_i$.

There are two possible starting points that a procedure can use in searching for a proof: the equational axioms in S or the inequations in S. If one used the former, then one would have to direct the application of the inference rules of R_ω, deriving conditional consequences from the equations in S, so as to arrive at a $p_i\rho \underset{s,t}{\sim} q_i\rho$ for some $p_i \neq q_i$ in S. While this seems to be a natural approach, it does not seem very effective for mechanical generation of proofs. The procedure described below uses the second approach.

Informally, the procedure can be summarized as follows. To prove $s \neq t$ as a consequence of a system S, we check if the two sides of an inequation in S can be transformed into a common term by applying "$\underset{s,t}{\sim}$-equivalent instantiations" to the

two sides. More formally, we search for a proof by deriving all possible *conditional inequations*

$$p \underset{skolem(s,t)}{\nsim} q$$

which are instances of the consequences of the inequations in S under the assumption $skolem(s = t)$. The procedure continues until we obtain a conditional inequation of the form

$$M \underset{skolem(s,t)}{\nsim} N$$

such that M and N are unifiable. (Note: Although in principle it is not necessary to skolemize s, t, the search space is substantially reduced by doing so.) A conditional inequation is deduced using a mechanism (which uses narrowing [15]) called $\underset{l,r}{\sim}$ *-narrowing* (defined below). This is somewhat similar to deriving inequations from S by paramodulating [28] from $l = r$ and the equations in S into the inequations in S.

Definition. A conditional inequation $p \underset{l,r}{\nsim} q$ or $q \underset{l,r}{\nsim} p$ is $\underset{l,r}{\sim}$ *-narrowed* by S to $p' \underset{l,r}{\nsim} q'$ or $q' \underset{l,r}{\nsim} p'$ (using σ) if p has a nonvariable subterm M such that *either* $p\sigma \underset{l,r}{\approx} p'$ and $q\sigma \equiv q'$, where σ is a substitution matching M with l or r, *or* $\exists m = n \in S$ (or $m \sim n \in S$) such that m and M unify with m.g.u. σ, $p' \equiv p[M \leftarrow n]\sigma$, and $q' \underset{l,r}{\equiv} q\sigma$.

Soundness of the $\underset{l,r}{\sim}$-narrowing is ensured, since each $\underset{l,r}{\sim}$-narrowing by S corresponds to replacing terms equated by the equations in S or by $l = r$. If there is a sequence of $\underset{l,r}{\sim}$-narrowing steps from $p \underset{l,r}{\nsim} q$, where $p \neq q$ is some inequation in S, to $m \underset{l,r}{\nsim} n$, then soundness ensures that $S \cup \{l = r\} \vDash m \neq n$.

procedure *ProveInequation*(s, t : *terms*; S : *system*);
  let σ be a skolem substitution for variables in s, t;
  $C := \{p \underset{s\sigma, t\sigma}{\nsim} q \mid p \neq q \in S\}$;
  while true do
    if $r \underset{s\sigma, t\sigma}{\nsim} r' \in C$ such that $\exists \rho . r\rho \equiv r'\rho$,
      then return with SUCCESS concluding $s \neq t$;
    let α be the oldest unnarrowed member of C;
    $C := C - \{\alpha\} \cup \{\xi \mid \alpha$ is $\underset{s\sigma, t\sigma}{\sim}$ -narrowed by S to $\xi\}$;
  end while
end procedure.

Example 5.1. The system given earlier in Example 4.4 is now used to illustrate procedure *ProveInequation*. This time, we prove that $0 \neq s(s(0))$ is a consequence of

the system S containing the following literals:

(a) $0 + x = x$,

(b) $s(x) + y = s(x + y)$,

(c) $x + y = y + x$,

(d) $x + (y + z) = (x + y) + z$,

(e) $x + (x + x) = 0$,

(f) $0 \neq s(0)$.

We give below the chain of proof steps, beginning from the inequation in S, and proceeding via a sequence of $\underset{0, ss0}{\sim}$-narrowing steps:

$$\frac{0 \neq s(0)}{0 \underset{0, ss0}{\sim} s(0)}$$

$$\frac{0 \underset{0, ss0}{\sim} s(0)}{0 \underset{0, ss0}{\sim} s(ss0)} \quad [\text{replacing } 0 \text{ by } ss0]$$

$$\frac{0 \underset{0, ss0}{\sim} s(ss0)}{0 \underset{0, ss0}{\sim} sss(0 + 0)} \quad [\sim\text{-narrowing by } 0 + x = x]$$

$$\frac{0 \underset{0, ss0}{\sim} sss(0 + 0)}{0 \underset{0, ss0}{\sim} ss(s0 + 0)} \quad [\sim\text{-narrowing by } s(x) + y = s(x + y)]$$

$$\frac{0 \underset{0, ss0}{\sim} ss(s0 + 0)}{0 \underset{0, ss0}{\sim} ss(0 + s0)} \quad [\sim\text{-narrowing by } x + y = y + x]$$

$$\frac{0 \underset{0, ss0}{\sim} ss(0 + s0)}{0 \underset{0, ss0}{\sim} s(s0 + s0)} \quad [\sim\text{-narrowing by } s(x) + y = s(x + y)]$$

$$\frac{0 \underset{0, ss0}{\sim} s(s0 + s0)}{0 \underset{0, ss0}{\sim} s(0 + (s0 + s0))} \quad [\sim\text{-narrowing by } 0 + x = x]$$

$$\frac{0 \underset{0, ss0}{\sim} s(0 + (s0 + s0))}{0 \underset{0, ss0}{\sim} s0 + (s0 + s0)} \quad [\sim\text{-narrowing by } s(x) + y = s(x + y)]$$

$$\frac{0 \underset{0, ss0}{\sim} s0 + (s0 + s0)}{0 \underset{0, ss0}{\sim} 0} \quad [\sim\text{-narrowing by } x + (x + x) = 0]$$

Since 0 unifies with 0, **return** with SUCCESS: $S \models 0 \neq ss0$.

5.1. *Completeness of Procedure ProveInequation*

Theorem 8. Procedure ProveInequation is complete for proving inequations, i.e., if $S \models s \neq t$, *then ProveInequation*(s, t, S) *returns with success.*

PROOF. Let $S \models s \neq t$, and let l, r represent the result of skolemizing terms s and t. The theorem results from the following observations:

(I) From the proof of Theorem 7, $(\exists \rho \exists p \neq q \in S) S \vdash_{R_\omega} (p\rho \underset{l,r}{\sim} q\rho)$.

(II) By Lemma 8.1 below, $p \underset{l,r}{\nsim} q$ can be $\underset{l,r}{\sim}$-narrowed by S in a finite number of steps to some $M \underset{l,r}{\nsim} N$ such that M and N are unifiable.

(III) The fair (breadth-first) strategy used by procedure *ProveInequation* guarantees that we eventually derive every conditional inequation obtainable by $\underset{l,r}{\sim}$-narrowing an inequation in S. □

To simplify the proof of observation (II) and the next lemma, we use a new "complex proof" tree structure, defined as follows.

Definition. A *complex proof* is a proof of $m \underset{l,r}{\nsim} n$ for some unifiable pair m, n, using $\underset{l,r}{\sim}$-narrowing steps as well as applications of inference rules in $R_\omega(l, r)$.

Lemma 8.1. If $S \vdash_{R_\omega} (p\rho \underset{l,r}{\sim} q\rho)$, *where* $p \neq q \in S$, *then there is a pair of unifiable terms* M, N *such that* $p \underset{l,r}{\nsim} q$ *can be* $\underset{l,r}{\sim}$*-narrowed by* S *in a finite number of steps to* $M \underset{l,r}{\nsim} N$.

PROOF. We start from a complex proof tree whose left subtree represents the proof $S \vdash_{R_\omega} p\rho \underset{l,r}{\sim} q\rho$, and right subtree contains only $p \underset{l,r}{\nsim} q$. The following transformation steps, each of which reduces the number of equations or $\underset{l,r}{\sim}$-literals in the complex proof tree, are applied at the root until the left subtree contracts into a single equation $\in S$. This process is then repeated for the right subtree, until all $\underset{l,r}{\sim}$-literals and equations $\notin S$ are eliminated from the complex proof tree. Finally, the complex proof tree represents an execution of *ProveInequation*, consisting entirely of $\underset{l,r}{\sim}$-narrowing steps issuing from some inequation in S, and resulting in some $m \underset{l,r}{\nsim} n$ with m, n unifiable, from which *ProveInequation* successfully deduces $l \neq r$.

We show below only the nontrivial transformations—deleting applications of E0 (reflexivity) and E1 (symmetry) is straightforward because the definition of the $\underset{l,r}{\sim}$-narrowing mechanism subsumes these rules. Applications of $\mathrm{E2}^{\omega}_{l,r}$ and $\mathrm{E4}^{\omega}_{l,r}$ are eliminated by transformations identical to those given below for E2 and E4 respectively, except that the symbol " $\underset{l,r}{\sim}$ " occurs instead of " = " in literals in the transformations. Similarly, applications of $\mathrm{E0}^{\omega}_{l,r}$ are eliminated by transformations identical to that given below for $\mathrm{E5}^{\omega}_{l,r}$, except that the symbol " $\underset{l,r}{\approx}$ " occurs instead of " = " in the relevant literals. We abbreviate " $\underset{l,r}{\nsim}$ " by " $\nsim$ " below.

of " $=$ " in the relevant literals. We abbreviate " $\underset{l,r}{\nsim}$ " by " $\nsim$ " below.

$$\text{(E2)}\ \frac{m=p,\ p=n}{m=n},\quad M \nsim N \ \Big/\ M\theta[m\theta \leftarrow n\theta] \nsim N\theta \quad \rightarrow\rightarrow \quad \frac{p=n,\ \dfrac{m=p,\ M \nsim N}{M\theta[m\theta \leftarrow p\theta] \nsim N\theta}}{M\theta[m\theta \leftarrow p\theta][p\theta \leftarrow n\theta] \nsim N\theta},$$

$$\text{(E3)}\ \frac{m=n}{m\rho=n\rho},\quad M \nsim N \ \Big/\ M\rho\theta[m\rho\theta \leftarrow n\rho\theta] \nsim N\rho\theta \quad \rightarrow\rightarrow \quad \frac{m=n,\ M \nsim N}{M\pi[m\pi \leftarrow n\pi] \nsim N\pi}$$

(where π is more general than $\rho\theta$),

$$\text{(E5}^{\omega}_{l,r}\text{)}\ \frac{\dfrac{m=n}{m \underset{l,r}{\sim} n},\quad M \nsim N}{M\theta[m\theta \leftarrow n\theta] \nsim N\theta}, \quad \rightarrow\rightarrow \quad \frac{m=n,\ M \nsim N}{M\theta[m\theta \leftarrow n\theta] \nsim N\theta},$$

$$\text{(E4)}\ \frac{p_1=q_1, \ldots, p_n=q_n}{f(\bar{p}_i)=f(\bar{q}_i)},\quad M \nsim N \ \Big/\ M\theta[f(\bar{p}_i)\theta \leftarrow f(\bar{q}_i)\theta] \nsim N\theta \quad \rightarrow\rightarrow$$

$$\frac{p_1=q_1,\ M \nsim N}{\dfrac{p_2=q_2,\ M\theta_1[p_1\theta_1 \leftarrow q_1\theta_1] \nsim N\theta_1}{p_3=q_3,\ M\theta_1[p_1\theta_1 \leftarrow q_1\theta_1]\theta_2[p_2\theta_2 \leftarrow q_2\theta_2] \nsim N\theta_1\theta_2}}$$

$$\vdots$$

$$\frac{p_n=q_n,\ M\theta_1[p_1\theta_1 \leftarrow q_1\theta_1] \cdots \theta_{n-1}[p_{n-1}\theta_{n-1} \leftarrow p_{n-1}\theta_{n-1}] \nsim N\theta_1 \cdots \theta_{n-1}}{M\theta_1[p_1\theta_1 \leftarrow q_1\theta_1] \cdots \theta_n[p_n\theta_n \leftarrow p_n\theta_n] \nsim N\theta_1\theta_2 \cdots \theta_n}$$

(where $(\theta_1\theta_2 \cdots \theta_n)$ is more general than θ, and each of the replacements $[p_i\theta_i \leftarrow q_i\theta_i]$ occurs within the subterm of M which originally unified with $f(\bar{p}_i)$).

□

5.2. *Towards a Decision Procedure*

We described above a semidecision procedure for the inherently undecidable problem of proving inequations. Using a complete *T-unification* procedure, it is possible to obtain a decision procedure for proving a restricted class of inequational consequences of some systems. If there is an algorithm for deciding T-unifiability of terms for the equational theory of $\mathrm{T} = \{skolem(s=t)\} \cup \{\text{equations in } S\}$, then we can decide whether or not $S \vDash s \neq t$. This is done by checking for each inequation in S whether its arguments can be T-unified. If some such inequation $p \neq q \in S$ has T-unifiable arguments p, q, then $\exists\rho.\mathrm{T} \vDash p\rho = q\rho$, which is inconsistent with $p \neq q \in S$; hence we decide that $S \vDash s \neq t$. Otherwise, if no inequation in S has T-unifiable arguments, we conclude that $skolem(s=t)$ is consistent with S; hence $s \neq t$ is not a logical consequence of S.

Hullot [15] has developed a sufficient criterion for the decidability of T-unification for an equational theory described by a finite set of oriented equations or "term rewriting system" (TRS). Intuitively, a narrowing is *basic* if each narrowing

step acts on (the reduct of) a term containing a nonvariable symbol of the original term from which narrowing began (see [15] for the technical definition). A TRS satisfies *Hullot's criterion* if it is canonical and if every basic narrowing derivation from the right hand side of every oriented equation terminates. There is a complete decision procedure for T-unification with respect to every equational theory T described by a TRS which satisfies Hullot's criterion. Based on this result, we have the following condition: we can decide whether or not $s \neq t$ is an inequational consequence of S if the equations in a system S together with $skolem(s = t)$ can be represented by a TRS which satisfies Hullot's criterion.

Some simple sufficient conditions can be given for Hullot's criterion to hold. For instance, we can decide whether $S \models s \neq t$ if equations in $S' \neq S \cup \{skolem(s = t)\}$ can be represented by a canonical TRS $\{lhs_i \rightarrow rhs_i\}$ such that each rhs_i is a ground term or a proper subterm of lhs_i [18]. For example, if S is an inequational system (without any equations), then, for any terms s, t, the equational theory of $\{skolem(s = t)\}$ can be represented by a canonical TRS in which each *rhs* is a ground term; hence the deduction of inequational consequences of S is decidable. Also, if $\{lhs_i \rightarrow rhs_i\}$ representing the equations in S is known to satisfy Hullot's criterion, and if neither s nor t unifies with any nonvariable subterm of any lhs_i, then there is a TRS satisfying Hullot's criterion with respect to the augmented theory (of equations in S'), and hence decidability of $S \models s \neq t$.

5.3. *Solving Inequations*

Frequently, an important question asked in logic programming is not whether a literal is a consequence of the theory, but whether the literal is "solvable" or "satisfiable", i.e., whether the system entails some instance of the given literal. For example, equations as well as inequations may have to be solved when using PROLOG-II or constraint logic programs [5, 16]. The problem of solving inequations in the empty theory has been addressed in earlier work [5, 19], as has the problem of solving equations in a theory (T-unification) [15]. The natural question which builds on these is the problem of solving inequations in a theory. Given a system S and an inequation $s \neq t$, we address the task of finding a substitution ψ such that $S \models s\psi \neq t\psi$.

We observe that a refutational proof method cannot be directly applied to this problem. Although a contradiction cannot be derived from $S \cup \{skolem(s = t)\}$, it may be the case that $S \models (s\psi \neq t\psi)$ for some substitution ψ. For example, $x \neq y$ is not a consequence of $\{p \neq q\}$, but it has a solution (substituting p for x and q for y); needless to say, $\{p \neq q\} \cup \{skolem(x = y)\}$ yields no contradiction.

The procedure we formulate for solving inequations is essentially derived from procedure *ProveInequation* given earlier. First, we rename all variables in all axioms of S to be distinct from each other and from variables in s, t. In the procedure, G is a set of two-tuples of conditional inequational consequences paired with the substitutions which have so far been accumulated for $s \neq t$. The new procedure *SolveInequation* differs from procedure *ProveInequation* in the following respects:

(1) the given terms s, t are not skolemized;

(2) conditional consequences are $\sim$-narrowed with the the *current* instantiation of $s = t$; and

(3) we keep track of the $\sim$ -narrowing substitutions successively made on $s = t$ at each stage, and the final answer returned is the composition of these substitutions.

We ignore substitutions which do not affect s, t —this is indicated in the last step of the procedure, where we distinguish between replacements of instances of terms equated by S and mutual replacements of instances of s, t. This does not affect the soundness and completeness of the procedure; any literal which is $\underset{x,x}{\sim}$ -narrowable is also $\underset{l,r}{\sim}$ -narrowable, for any l, r.

procedure *SolveInequation* $(s, t : terms; S : System)$;
$G := \{\langle p \nsim q, identity\rangle | p \neq q \in S\}$;
while true do
  if $\langle r \nsim r', \psi\rangle \in G$ such that r and r' unify with m.g.u. ρ,
    then return successfully with the solving substitution $\psi\rho$;
  let $\langle m \nsim n, \sigma\rangle$ be the oldest unnarrowed member of G;
  $G := (G - \{\langle m \nsim n, \sigma\rangle\})$
      $\cup\{\langle p \nsim q, \sigma\rangle | m \nsim n$ is $\sim$ -narrowed by S to $p \nsim q\}$
      $\cup\{\langle p \nsim q, \sigma\rho\rangle | m \nsim n$ is $\underset{s\sigma, t\sigma}{\overset{x,x}{\sim}}$ -narrowed by the empty set to $p \nsim q$ using $\rho\}$;
end while
end procedure.

Proposition 4 (Soundness). If procedure SolveInequation (s, t, S) returns with a substitution ψ, then $S \models s\psi \neq t\psi$.

PROOF. Straightforward, by showing that $S \cup \{skolem(s\psi = t\psi)\} \vdash_{R_0} \square$. $\square$

Proposition 5 (Completeness). If there exists ψ such that $S \models s\psi \neq t\psi$, then SolveInequation (s, t, S) successfully returns with some substitution.

PROOF. Follows from the completeness of procedure *ProveInequation*: since $S \models [s\psi \neq t\psi]$, S contains some inequation from which $s\psi \neq t\psi$ can be derived by $\underset{s\psi, t\psi}{\sim}$ -narrowing. Since any literal which is $\underset{s\psi, t\psi}{\sim}$ -narrowable is also $\underset{s, t}{\sim}$ -narrowable, *SolveInequation* also terminates successfully. $\square$

6. CONCLUDING REMARKS

In this paper, we have explored reasoning in systems of equations and inequations, giving special emphasis to developing a forward reasoning method for proving inequational consequences. We developed an inference set R_ω and proved its forward completeness. We used the completeness result to construct goal-directed semidecision procedures for proving and solving inequations. Proving an inequational consequence $s \neq t$ is decidable if the equations in the system together with $skolem(s = t)$ meet the criteria for the decidability of T-unification.

The expressive power of declarative languages will be enhanced if inequations can be stated explicitly. To develop evaluation strategies for such programming and specification languages, it is important to study forward reasoning techniques for checking inequations. We believe that the results developed above can serve as

a basis for developing better strategies and decision procedures for proving and solving inequations. For practicability and efficiency, however, our results need to be extended to systems containing equations specified as canonical TRS.

In R_ω-proofs, unlike refutational proofs, no new equations are introduced into the system. Every formula m $\underset{l,r}{\sim}$ n derived at an intermediate stage is a meaningful conditional consequence of the system, unlike formulas obtained during refutational proofs. Whenever there is an R_0-refutation proof, there is a forward proof tree of the same or smaller size using R_ω. Since there is no transitivity inference rule between the relations $\neq$ and $\underset{s,t}{\sim}$, the search space—the number of minimal R_ω-proofs of a literal from a given system—is no greater (generally less) than that of refutational proofs. However, in contrast with forward proof techniques, refutational theorem-proving techniques have already been applied and implemented using "well-behaved" term-rewriting systems. The same must be explored for proofs in R_ω in order to get a good comparison of the two approaches.

Selman [29] gives an inference system which is forward complete for deducing every "equational implication" $\wedge_i a_i = b_i \supset c = d$ which is a consequence of a finite set S of equational implications. If an inequation $p \neq q$ is construed to be equivalent to $p = q \supset \mathbf{true} = \mathbf{false}$, then this implication is deducible if it is a consequence of S. However, since Selman is concerned with a more general formalism than just equations and inequations, his inference rules are much more complex, and include generalized modus ponens, arbitrary introduction of antecedents, and transitivity of implication. We also observe that there is an implicit inequational axiom **true** $\neq$ **false**, on which such an inequational proof relies.

Our procedure for solving inequations can be of importance for constraint logic programming, expanding the language of allowable constraints to theories specified by systems of equations and inequations. Our results about the structure of inequational proofs, establishing that any inequational consequence has a proof using only inequations in the given system, are also important. The significance of the independence of negative constraints has been pointed out in [22] for the design of faster and parallel algorithms for solving and checking constraints, especially when literals can be evaluated to a canonical form.

APPENDIX

This appendix contains the complete proof of Theorem 4, and the proofs of the lemmas needed to prove Theorem 3. It also contains the proof of Proposition 1.

A.1. Proof of Theorem 4

Theorem 4 (Refutational completeness of R_0). If $S \models s \neq t$, then

$$S \cup \{skolem(s = t)\} \vdash_{R_0} \square .$$

PROOF. If $S \models s \neq t$, then $S \cup \{s\sigma = t\sigma\}$ is unsatisfiable, where σ is a skolemizing substitution for s and t. Therefore, by Lemma 4.1 (below), $\square$ can be derived from $S \cup \{s\sigma = t\sigma\}$ using "positive hyperresolution" and "positive hyperparamodulation" (described below for the case where each clause is a literal; see [4] for the

precise definition) together with the functionally reflexive axioms $f(\bar{t}_i) = f(\bar{t}_i)$ for every symbol f and all terms t_i. We show that every proof using these axioms and inference rules on equations and inequations has an equivalent proof using the inference rules of R_0. We conclude that R_0 is refutationally complete because every inference that can be accomplished using a known refutationally complete strategy can also be achieved using the rules in R_0.

The functionally reflexive axioms (instances of $x = x$) can be obtained simply by invoking rule E0. In systems consisting of equations and inequations, paramodulation and resolution derive only single-literal clauses. Resolution between equations and inequations is just the transitivity (I2) step preceding the final refutation: a proof of some $p \neq p$ from an equation and an inequation, possibly after first instantiating and/or applying the symmetry rule to the literals resolved.

In this proof, we abbreviate $m[t \leftarrow s]$, $n[t \leftarrow s]$, $m[t \leftarrow r]$, and $n[t \leftarrow r]$ by $m[s]$, $n[s]$, $m[r]$ and $n[r]$ respectively. Positive hyperparamodulation between equations and inequations is the inference rule

$$\frac{m[t] = n[t],\ r = s,\ t\sigma \equiv r\sigma}{m[s]\sigma = n[s]\sigma},$$

where σ is the m.g.u. of t and r. This has the following equivalent proof using R_0:

$$\dfrac{\dfrac{\dfrac{r = s}{s = r}\,(E1)}{m[s] = m[r]}\,(E4) \qquad \dfrac{m[t] = n[t]}{m[t]\sigma = n[t]\sigma}\,(E3) \qquad \dfrac{r = s}{n[r] = n[s]}\,(E4)}{m[s]\sigma = n[t]\sigma}\,(E2)^1 \qquad \dfrac{}{n[r]\sigma = n[s]\sigma}\,(E3)$$

$$\dfrac{}{m[s]\sigma = n[s]\sigma}\,(E2)^1 \qquad \square$$

Lemma 4.1. For an E-unsatisfiable set of clauses S, there is a proof of $\square$ *from* $S \cup \{x = x\} \cup F$, *using inference rules of positive hyperresolution and hyperparamodulation, where F is the set of functionally reflexive axioms for S.*

PROOF. See [4].

A.2. Lemmas Used in the Proof of Theorem 3

In the following we give a series of lemmas (along with their proofs), each of which essentially gives a set of transformations necessary to get an R_0-proof tree into a desired form. These transformations were used in the proof of Theorem 3. To assist in reasoning about transformations of proof subtrees, we consider, besides R_0, the inference sets $R_{cons} = \{E1, I1, E2, I2, E3, I3\}$ and $R_{trans} = \{I0, E1, I1, E2, I2\}$;

[1] (E2) is applicable because $m[r]\sigma \equiv m[t]\sigma$ and $n[t]\sigma \equiv n[r]\sigma$.

proofs using these subsets of R_0 are referred to as *conservative* and *transitive* proofs respectively.

Lemma 3.1 (Duality between equational and inequational proof trees). Let P be a conservative (or transitive) proof tree of an equation $l_0 = r_0$. Let $l_0 = r_0, \ldots, l_n = r_n$ be any chain of equations from the root to a leaf of P. Let P' be a tree obtained by replacing in P every equation $l_i = r_i$ $(0 \leq i \leq n)$ in the chain by the inequation $l_i \neq r_i$. Then P' is also a conservative (or transitive) proof tree.

PROOF. From Proposition 2.1 (the independence result for equational deduction), it follows that proofs of equations in R_{cons}, R_{trans} do not use inequations at any stage. For each step in the proof tree at which $l_i \neq r_i$ is substituted for $l_i = r_i$, the new inference remains valid: instances of E1, E2, and E3 become respectively transformed to instances of I1, I2, and I3. Other inferences are untouched. So the transformed tree continues to represent a proof in R_{cons} or R_{trans} respectively. However, the new proof uses axioms (leaves in the proof tree) from a different system, and derives a different theorem than the original proof. □

NOTE. The above argument does not hold for the subterm inference rule (E4). Hence the result is not true for R_0-proofs.

Lemma 3.2. In the proof tree of an inequation using inference rules from R_0 (or R_{trans} or R_{cons}), if all inequations $l_i \neq r_i$ are replaced by the corresponding equations $l_i = r_i$, the resulting structure continues to be a proof tree using inference rules from R_0 (or R_{trans} or R_{cons}, respectively).

PROOF. Similar to the above: applications of inference rules I1–I3 are respectively replaced by applications of E1–E3.

Lemma 3.3. If $A \vdash_{R_0} p \neq q$, then there is a proof of $p \neq q$ (from A using R_0) in which applications of transitivity inference rules (E2, I2) *are not followed by the applications of any other inference rules.*

PROOF. The proof of this lemma is almost identical to the proof of Lemma 3.5 for equational derivations, and relies on applying the following transformations, followed by those shown for Lemma 3.5:

$$(\mathrm{I1})\ \frac{(\mathrm{I2})\ \dfrac{q \neq p,\ p = r}{q \neq r}}{r \neq q} \quad \to\to \quad (\mathrm{I2})\ \frac{(\mathrm{E1})\ \dfrac{p = r}{r = p} \quad (\mathrm{I1})\ \dfrac{q \neq p}{p \neq q}}{r \neq q},$$

$$(\mathrm{I3})\ \frac{(\mathrm{I2})\ \dfrac{q \neq p,\ p = r}{q \neq r}}{q\sigma \neq r\sigma} \quad \to\to \quad (\mathrm{I2})\ \frac{(\mathrm{I3})\ \dfrac{q \neq p}{q\sigma \neq p\sigma}, \quad (\mathrm{E3})\ \dfrac{p = r}{p\sigma = r\sigma}}{q\sigma \neq r\sigma}. \qquad \square$$

Lemma 3.4. Let P be a literal at the leaf of a transitive proof tree T deriving $s \neq s$. Then T can be transformed to another transitive proof tree which has the same leaves as T, but has P as a premise in the final proof step, deriving $p \neq p$ for some term p.

PROOF. Consider the transitive proof tree of $s \neq s$ containing one or more applications of rules E1, E2, I1, I2. By Proposition 2.2, there is only one inequation two

hops above the root; we show that this inequation can be brought down one level, resulting in another transitive proof tree of $s \neq s$ with the same leaves. If I1 (symmetry) was the rule applied to this inequation, then E1 may instead be applied to the equation at its "uncle" node (parent's sibling) in the tree. If I2 (transitivity) was the rule applied, then E2 may instead be applied to the equations at its sibling and uncle nodes:

$$(\mathrm{I2})\ \frac{(\mathrm{I2})\ \dfrac{s \neq t,\ t = r}{s \neq r},\quad r = s}{s \neq s} \quad \rightarrow\rightarrow \quad (\mathrm{I2})\ \frac{s \neq t,\quad (\mathrm{E2})\ \dfrac{t = r,\ r = s}{t = s}}{s \neq s},$$

$$(\mathrm{I2})\ \frac{(\mathrm{I1})\ \dfrac{t \neq s}{s \neq t},\quad t = s}{s \neq s} \quad \rightarrow\rightarrow \quad (\mathrm{I2})\ \frac{t \neq s,\quad (\mathrm{E1})\ \dfrac{t = s}{s = t}}{t \neq t}.$$

By repeating these tree transformations, a leaf inequation can be brought down until it is just above the root in the transitive proof tree. By Lemma 3.2, similar transformations can be applied for lowering equations, i.e., a leaf equation in a transitive proof of $s \neq s$ can be lowered until it is a child of the root. □

Lemma 3.5. The R_0-proof tree τ of any equation can be transformed into another R_0-proof tree τ', in which applications of transivity (E2) *are not followed by the applications of any other inference rule.*

PROOF. By Proposition 2, τ contains only applications of rules E0–E4. E0 can be applied only at the leaves of a tree, and can hence be ignored. We repeatedly transform τ, pulling down the applications of E2 towards the final result and away from the leaves. After repeating these transformations as often as possible, we obtain the desired proof tree with a transitive proof subtree containing the root; only the part of the proof tree closest to the conclusion contains applications of transitivity (E2):

$$(\mathrm{E1})\ \frac{(\mathrm{E2})\ \dfrac{q = p,\ p = r}{q = r}}{r = q} \quad \rightarrow\rightarrow \quad (\mathrm{E2})\ \frac{(\mathrm{E1})\ \dfrac{p = r}{r = p},\quad (\mathrm{E1})\ \dfrac{q = p}{p = q}}{r = q},$$

$$(\mathrm{E3})\ \frac{(\mathrm{E2})\ \dfrac{q = p,\ p = r}{q = r}}{q\sigma = r\sigma} \quad \rightarrow\rightarrow \quad (\mathrm{E2})\ \frac{(\mathrm{E3})\ \dfrac{q = p}{q\sigma = p\sigma},\quad (\mathrm{E3})\ \dfrac{p = r}{p\sigma = r\sigma}}{q\sigma = r\sigma},$$

$$(\mathrm{E4})\ \frac{s_1 = t_1, \ldots,\quad (\mathrm{E2})\ \dfrac{s_k = r,\ r = t_k}{s_k = t_k},\ \ldots,\quad s_n = t_n}{f(s_1, \ldots, s_k, \ldots, s_n) = f(t_1, \ldots, t_k, \ldots, t_n)} \quad \rightarrow\rightarrow$$

$$(\mathrm{E2})\ \frac{(\mathrm{E4})\ \dfrac{s_1 = t_1, \ldots, s_k = r, \ldots, s_n = t_n}{f(\bar{s}_i) = f(t_1, \ldots, r, \ldots, t_n)},\quad (\mathrm{E4})\ \dfrac{(\mathrm{E0})\ \dfrac{}{t_1 = t_1}, \ldots,\quad r = t_k, \ldots,\quad (\mathrm{E0})\ \dfrac{}{t_n = t_n}}{f(t_1, \ldots, r, \ldots, t_n) = f(\bar{t}_i)}}{f(s_1, \ldots, s_k, \ldots, s_n) = f(t_1, \ldots, t_k, \ldots, t_n)}.$$

□

A.3. PROOF OF PROPOSITION 1

Proposition 1. The proof tree of any equation can be transformed into another proof tree in which applications of transitivity (E2) *are not followed by those of any other*

inference rule, and applications of the superterm rule (E4) *are followed only by transitivity* (E2).

PROOF. In addition to the transformations shown in the proof of Lemma 3.5, we also apply the following transformations to pull down applications of (E4) towards the root of the proof tree:

$$\begin{array}{l} \dfrac{\dfrac{s1 = t1, \ldots, sn = tn}{f(s1 \ldots sn) = f(t1 \ldots tn)}\,(E4)}{f(t1 \ldots tn) = f(s1 \ldots sn)}\,(E1) \end{array} \quad \rightarrow\rightarrow \quad \begin{array}{l} \dfrac{\dfrac{s1 = t1, \ldots, sn = tn}{t1 = s1 \quad \ldots \quad tn = sn}\,(E1)}{f(t1 \ldots tn) = f(s1 \ldots sn)}\,(E4) \end{array}$$

$$\begin{array}{l} \dfrac{\dfrac{s1 = t1, \ldots, sn = tn}{f(s1 \ldots sn) = f(t1 \ldots tn)}\,(E4)}{f(s1\sigma \ldots sn\sigma) = f(t1\sigma \ldots tn\sigma)}\,(E3) \end{array} \quad \rightarrow\rightarrow \quad \begin{array}{l} \dfrac{\dfrac{s1 = t1 \quad \ldots \quad sn = tn}{s1\sigma = t1\sigma \quad \ldots \quad sn\sigma = tn\sigma}\,(E3)}{f(s1\sigma \ldots sn\sigma) = f(t1\sigma \ldots tn\sigma)}\,(E4) \end{array}$$

□

REFERENCES

1. Aiba, A. and Sakai, K., CAL: A Theoretical Background of Constraint Logic Programming and its Applications, to appear.
2. Birkhoff, G., On the Structure of Abstract Algebras, *Proc. Cambridge Philos. Soc.* 31:433–454 (1935).
3. Boyer, R. S., and Moore, J. S., *A Computational Logic*, Academic, New York, 1979.
4. Chang, C. L. and Lee, R. C., *Symbolic Logic and Mechanical Theorem Proving*, Academic, New York, 1973.
5. Colmerauer, A., Equations and Inequations on Finite and Infinite Trees, in: *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT Japan, 1984.
6. Comon, H., Sufficient Completeness, Term Rewriting Systems, and "Anti-unification", in: *Proceedings of the 8th Conference on Automated Deduction*, Oxford, LNCS 230, Springer-Verlag, July 1986.
7. Dershowitz, N., Equations as Programming Language, in: *Proceedings of the 4th Jerusalem Conference on Information Technology (JCIT)*, IEEE, 1984.
8. Dershowitz, N. and Plaisted, D. A., Logic Programming *cum* Applicative Programming, in: *Proceedings of the 1985 Symposium on Logic Programming*, Boston, 1985.
9. Futatsugi, K., Goguen, J., Jouannaud, J. P., and Meseguer, J., Principles of OBJ2, in: *Proceedings of the ACM Symposium on Principles of Programming Languages—12*, 1985, pp. 52–66.
10. Gentzen, G., Untersuchungen über das Logische Schliessen, *Math. Z.* 39:176–210, 405–431 (1934–35).
11. Goguen, J. A. and Meseguer, J., Equality, Types, Modules and Generics for Logic Programming, *J. Logic Programming* 1(2):179–210 (1984).
12. Guttag, J. V. and Horning, J. J., The Algebraic Specification of Abstract Data Types, *Acta Inform.* 10:27–52 (1978).
13. Hoffman, C. M. and O'Donnell, M. J., Programming with Equations, *ACM Trans. Prog. Lang. and Systems* 4(1):83–112 (Jan. 1982).
14. Hsiang, J. and Rusinowitch, M., On Word Problems in Equational Theories, in: *Proceedings of 14th ICALP*, Karlsruhe, West Germany, July 1987.

15. Hullot, J. M., Canonical Forms and Unification, in: *Proceedings of the 5th Conference on Automated Deduction*, Lecture Notes in Comput. Sci. 87, Springer-Verlag, Apr. 1980.
16. Jaffar, J. and Lassez, J. L., Constraint Logic Programming, Tech. Rep., IBM T. J. Watson Research Center, Yorktown Heights, N.Y., June 1986; also in *ACM Symposium on Principles of Programming Languages—'87*.
17. Jaffar, J. and Michaylov, S., Methodology and Implementation of a CLP System, in: *Proceedings of the 1987 Logic Programming Conference*, Melbourne, Australia, 1987.
18. Kapur, D., Narendran, P., and Otto, F., On Ground Confluence of Term Rewriting Systems, Technical Report 87-06, GE Corp. Res. and Dev. Center, Schenectady, NY 12345, 1987.
19. Kirchner, C. and Lescanne, P., Solving Disequations, in: *Proceedings on the 2nd IEEE Symposium on Logic on Computer Science*, 1987.
20. Kleene, S. C., *Introduction to Metamathematics*, Van Nostrand, 1952.
21. Lassez, J. L., Maher, M. J., and Marriott, K., Unification Revisited, in: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan-Kaufmann, 1987.
22. Lassez, J. L. and McAloon, K., Independence of Negative Constraints, unpublished manuscript, IBM T. J. Watson Research Center, May 1988.
23. Lassez, J. L. and McAloon, K., Applications of a Canonical Form for Generalized Linear Constraints, FGCS-88, Tokyo, Japan, 1988.
24. Mohan, C. K. and Srivas, M. K., Conditional Specifications Using Inequational Assumptions, in: *Proceedings of the First International Workshop on Conditional Term Rewriting Systems*, Univ. of Paris-Sud, Orsay, France, LNCS 308, Springer-Verlag, July 1987.
25. Mohan, C. K., Srivas, M. K., and Kapur, D., Reasoning in Systems of Equations and Inequations, in: *Proceedings of the 7th Conference on Foundations of Software Technology and Theoretical Computer Science (7FST&TCS)*, Pune, India, LNCS 287, Springer-Verlag, Dec. 1987, pp. 305–325.
26. Mohan, C. K., Negation in Equational Reasoning and Conditional Specifications, Ph.D. Thesis, State Univ. of New York at Stony Brook, 1988.
27. Musser, D. R., Abstract Data Type Specification in the AFFIRM System, in: *Proceedings of the Specifications of Reliable Software Conference*, Boston, Apr. 1979, pp. 47–57.
28. Robinson, G. A. and Wos, L., Paramodulation and Theorem Proving in First Order Theories with Equality, in: B. Meltzer and D. Michie (eds.), *Machine Intelligence IV*, American Elsevier, New York, 1969, pp. 135–150.
29. Selman, A., Completeness of Calculii for Axiomatically Defined Classes of Algebras, *Algebra Universalis* 2:20–32 (1972).
30. Wos, L. and McCune, W., Negative Paramodulation, in: *Proceedings of the 8th Conference on Automated Deduction (CADE-8)*, Oxford, LNCS 230, Springer-Verlag, July 1986.
31. Zhang, H. and Kapur, D., First-Order Theorem Proving Using Conditional Rewrite Rules, in: *Proceedings of the 9th International Conference on Automated Deduction*, Argonne, Ill., LNCS 310, Springer-Verlag, May 1988, pp. 1–20.

REPRESENTING KNOWLEDGE WITH THEORIES ABOUT THEORIES

EDWARD P. STABLER, JR.

▷ Theories about proofs in other theories can be used not only to provide representations of completed proofs, but also to provide an elegant, declarative, and logically pure method for controlling deductions. This idea is used implicitly in a widening range of applications, and deserves explicit consideration. In this paper, a technique for generating a standard proof-representation-building metatheory for Horn-clause theories defined, its logical semantics is carefully considered, and the sense in which the technique is correct and complete is defined. Then we show how such metatheories can elegantly represent a wide range of problems. We focus on some problems which are naturally formulated in terms of overly general axioms together with conditions on proofs which block exactly the derivations of incorrect results: diagnosis, planning, and natural-language parsing. This surprising approach can yield representations that are succinct, feasible, and close to the most intuitive, informal statement of the problem. Methods for using such an approach efficiently with left-to-right theorem provers are described. ◁

1. INTRODUCTION

Theories about sentences or proofs in another theory are often useful. Some theories of knowledge, belief, and action, for example, use representations of the sentences believed by an agent [20]. In the most straightforward logical approaches to parsing, the output is a proof tree [24]. In other applications, a proof tree is taken as input to further processing. For example, a representation of a proof is

Address correspondence to Dr. E. P. Stabler, Department of Linguistics, University of California, Los Angeles, California 90024-1543.

Received August 1987; accepted October 1988.

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/90/$3.50

sometimes displayed as a justification of proven results, or it is used by programs that aid in detecting errors in the axioms of the object theory [31]. The focus of this paper, though, will be on applications that use restrictive conditions on proofs: conditions that are not satisfied by any proof of certain provable entailments of the object theory. These conditions, we will say, "sacrifice the completeness" of the proof method of the object theory. Our approach to defining conditions on proofs differs from many others in that these conditions are defined in first-order logic as relations over provably correct representations of first-order resolution proofs. The advantages of using a declarative, first-order logical representation, with its well-understood semantics and proof techniques, can thus be carried over to the metatheoretic level.

The most common application of conditions on proof trees is in controlling the search for a refutation by pruning the derivation tree being searched. This technique is to be distinguished from other "metalevel" strategies that only control the order in which the tree is searched. Some of the former, pruning techniques are well known. For example, some proofs can be eliminated from a resolution search space without affecting the completeness of the proof procedure. One of the best-known strategies of this sort is the elimination of "repetitive" derivations, i.e., derivations in which a clause identical to one of its ancestors is derived by input resolution [17, 29, 6].

In other systems, the search for a proof is controlled in ways that may sacrifice completeness, although that is not their goal. For example, heuristic techniques, the imposition of bounds on the depth or breadth of a proof, and interactive theorem provers may restrict the search for a proof in ways that sacrifice completeness, but their goal is just to avoid unnecessary search. There is a third class of applications for restrictions on proofs, though, which are naturally expressed in terms of axioms together with conditions on the proofs that *deliberately* sacrifice completeness. In these cases, the set of refutations from the axioms is a superset of the set of desired, "correct" refutations. Strictly speaking, under the usual interpretation, the axioms are overly general, but the incorrect entailments are never proven because conditions on the proofs block the proof of exactly those results. Another way to look at the matter is to view the basic axioms as correct characterizations of the set of possible solutions, where the correct solutions are then characterized as possible solutions whose proofs in the original theory have special properties. A simple logical foundation and applications of this approach will be the focus of this paper.

This basic idea was pioneered by Chomsky [8] in his use of multiple levels of representation in the description of natural languages: a basic theory (a generative grammar) characterizes a class of strings, and then a second level of principles applies to the proofs that those strings follow from the basic theory (i.e. to derivations from the "base" grammar). This sort of strategy is still discernible in the most recent efforts in Chomskian syntax. A similar problem representation seems to be valuable in other domains as well.[1] The application of a metatheoretic

[1]Although it was parsing techniques inspired by Chomskian syntax that inspired this work, the connection between the problem representations described here and Chomsky's theories of language is rather loose. Chomsky defined transformations on the "base" trees and principles that must be satisfied by the outputs of these transformations. The well-formedness conditions on the outputs of the transformations, then, only indirectly restrict the set of "base" trees that can correspond to good sentences.

representational strategy to parsing with Chomskian concepts is only one of the natural applications of metatheoretic strategies.

We focus on Horn-clause problems because they allow such simple proof techniques and fast implementations. The extension of the approach to full first-order logic, though, is straightforward. In this paper, we consider some problems that are naturally expressed in terms of Horn-clause axioms together with constraints on proofs, and then we elaborate our metatheoretic approach to obtain correct representations that are more feasibly managed by left-to-right Horn-clause refutation systems like PROLOG, SLD resolution with a leftmost selection rule [16], or Earley deduction.[2] Finally, we will contrast our approach with related work and consider directions for further research.

2. LOGICAL FOUNDATIONS

To constrain derivations we must first have terms to represent them. There are standard techniques for defining proof trees of a theory. We will begin with one of the simplest techniques. We transform a theory S into a theory $\tau_0(S)$ that specifies derivation trees in S.

In the presentation of his incompleteness theorem, Gödel formalized derivations by assigning natural numbers to the primitive symbols, thus representing formulas by finite sequences of natural numbers and proofs by finite sequences of finite sequences of natural numbers [12]. It is illuminating to note some of the respects in which the present approach differs from Gödel's. In the first place, we will not use a system in which a proof or any other expression can contain a term that, under the intended interpretation, refers to an expression properly containing it. Thus no predication can assert anything about itself. Under our intended interpretation, terms of $\tau_0(S)$ have exactly the same length and the same structure as the terms of S that they refer to: the interpretation of these terms is given by a symbol for symbol bijection. This makes the semantics and the proofs in the metatheory $\tau_0(S)$ extremely simple. But it is clear that this same property makes self-referential formulas impossible in a standard logic, for notice that we immediately have the consequence that no finite expression can properly contain a name of itself. A second difference between our approach and standard Gödel numbering in arithmetic is that we do not need to define a representation of proofs in S in the language of S itself; rather, it is convenient to allow the language of $\tau_0(S)$ to be a simple transformation of the language of S. These differences allow us to use an approach that is more concise and more efficient than the one Gödel needed. One final difference of less importance is that we represent proofs with trees rather than with sequences: trees provide useful information about the structure of the proof.

We will first define a syntactic transformation τ_0 that maps a theory S into a different theory $\tau_0(S)$, which specifies the proof trees of S. We then specify the intended semantic interpretation of the output of this transformation, the range of τ_0. And finally we establish some of the important properties of τ_0.

[2] Earley deduction, an "all paths at once" resolution strategy inspired by Earley's context-free parsing algorithm, is described in [25].

2.1. *The Syntactic Transformation*

We begin by motivating our approach with a simple example. The function τ_0 transforms a logical theory expressed in Horn clauses into another theory which defines derivations in the original theory. The basic idea of the transformation is really quite simple and well known: the proof is represented by a variable added as a new argument to every predicate in the body of a clause, and by a term added as argument to every predicate in the head of a clause, as illustrated in the following example:[3]

$$
\begin{aligned}
S = \{ \quad & \leftarrow man(x), \\
& man(x) \leftarrow male(x), human(x), \\
& male(x) \leftarrow , \\
& human(x) \leftarrow \quad \}, \\
\tau(S) = \{ \quad & \leftarrow man(x, Proof), \\
& man(x, man(x)/[Q, R]) \leftarrow male(x, Q), human(x, R), \\
& male(x, male(x)) \leftarrow , \\
& human(x, human(x)) \leftarrow \quad \}.
\end{aligned}
$$

Notice that one instance of $\leftarrow man(x, Proof)$ that is inconsistent with the other clauses in $\tau_0(S)$ is one in which x remains uninstantiated and *Proof* is instantiated to the term $man(x)/[male(x), human(x)]$. Letting the slash represent the dominance relation and putting the subtrees under a node in a list, this term represents the tree with a root labeled $man(x)$ dominating two leaves labeled $male(x)$ and $human(x)$, respectively, as shown in Figure 1. We interpret this as a proof tree indicating that, in S, $\forall man(x)$ can be proven by proving $\forall male(x)$ and $\forall human(x)$, which in turn can be proven directly from unit clauses. We now explain these matters in detail, beginning with the syntax.

Consider our previous example. Notice that language of $\tau_0(S)$ is not the same as the language of S: it includes new predicates, and a number of new function symbols: the unary function symbols *man*, *male*, and *human*, the binary (infix) function /, and the list functions (which are, in canonical form, · and [ ]). We will define the language L' of $\tau_0(S)$ in terms of L as follows.

Let L be the language of an arbitrary theory S. Define a bijection *name* from the n-ary ($n \geq 0$) predicate symbols of L onto a set of n-ary function symbols that do not occur in L. Then the *the set of function symbols of* L' is the union of the

[3] We follow the terminology and notation of [16], except that we will use not only 'x','y','z',... as names of variables, but any symbol beginning with an uppercase letter or underscore, as is done in many PROLOGs.

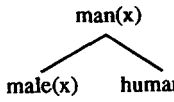

FIGURE 1. A simple proof tree.

function symbols of L with the set F of new n-ary function symbols,

$$F = \{f^n | f^n = name(P^n) \text{ for some predicate } P^n \text{ in } L\},$$

together with a set T containing three new function symbols (whose arity we have indicated with superscripts):

$$T = \{.^2, [\,]^0, /^2\}.$$

The first two symbols in T are the standard binary "list constructor" and the 0-ary "empty list;" if these functors already occur in S, different symbols can be used to avoid ambiguity. To simplify the presentation, we assume that the symbols in T do not occur in L or F. And we assume that L does not use any symbol as an n-ary function symbol and as an n-ary predicate symbol, so *name* can be a syntactic identity function. As is standard, we adopt the notational convention of using the more readable bracket notation for lists (e.g., $[a,[b]]$) as an abbreviation for the canonical functional notation $(.(a,.(.(b,[\,]),[\,]))$ or infix notation $a.((b.[\,]).[\,]))$.

The *set of predicate symbols of* L' is the set (again using superscripts to indicate arity)

$$\{P^{n+1} | P^n \text{ a predicate in } L\}.$$

Definition of τ_0.

(i) For any set S of Horn clauses, $\tau_0(S) = \{\tau_0(C) | C \in S\}$.

(ii) For any Horn clause C such that for some $k > 0$

$$C = \leftarrow A_1, \ldots, A_k,$$

we define

$$\tau_0(C) = \leftarrow A'_1, \ldots, A'_k,$$

where if the predicate of A_i is an n-ary predicate P ($n \geq 0$), then A'_i is the $n+1$-ary predicate P with the n arguments of A_i as its first arguments and with a new variable as its last argument.

(iii) For any Horn clause C such that

$$C = A_0 \leftarrow,$$

we define

$$\tau_0(C) = A'_0 \leftarrow,$$

where the predicate of A_0 is an n-ary predicate P ($n \geq 0$), and the predicate of A'_0 is the $n+1$-ary predicate P with the n arguments of A_0 as its first arguments and with (the function expression that is syntactically identical to) A_0 itself as its last argument.

(iv) For any Horn clause C such that for some $k > 0$

$$C = A_0 \leftarrow A_1, \ldots, A_k,$$

we define

$$\tau_0(C) = A'_0 \leftarrow A'_1, \ldots, A'_k,$$

where

(a) if the predicate of A_i $(k \geq i \geq 1)$ is an n-ary predicate P $(n \geq 0)$, then A'_i is the $n + 1$-ary predicate P with the n arguments of A_i as its first arguments and with a new variable as its last argument, and

(b) if the predicate of A_0 is an n-ary predicate P $(n \geq 0)$, then A'_0 is the $n + 1$-ary predicate P with the n arguments of A_0 as its first arguments and with $A_0/[V_1, \ldots, V_k]$ as its last argument, where $V_1, \ldots, V_k$ are the k new variables introduced by step (a).

Notice that according to this definition, τ_0 is not really functional, since the choice of "new variables" is not determined by the source clause. However, it is clear that all the values of $\tau_0(C)$ for any C are variants of one another: since we are interested in the entailments of the values of $\tau_0(S)$, any one of these values will suffice.

One other point that should be noted is that τ_0 introduces some redundancy: the representation of the proof repeats information that is specified in other arguments of the predicates. Obviously, we could eliminate some of this redundancy in various ways. The reason for keeping the redundancy in τ_0 is, intuitively, just that it keeps the "tree-building" part of the theory separate from the arguments needed to define the entailments. As a result, this formulation is slightly more convenient in the proof of the "noninterference" property of τ_0, defined below. This formulation also facilitates the consideration of sound ways of building something less, or something rather different, than complete proof trees, as discussed in the section below on efficient representations.

2.2. *An Appropriate Semantic Interpretation of the Transformed Theory*

As noted above, transformations like τ_0 are really very common in logic programming, but they are often introduced as proof-theoretic tricks, with little semantic explanation. The basic idea we will use has been well explained, though, as in the following passage by Moore:

> Typically this sort of thing is done using string operations like concatenation, so that the conjunction of P and Q would be represented by something like '('|P|' $\wedge$ '|Q|')'. ... There is a much more elegant way to do the encoding, however, which is due to McCarthy (1962) [19]. For purposes of semantic interpretation of the object language, which is what we want to do, the details of the syntax are largely irrelevant. In particular, the only thing that we need to know about the syntax of conjunctions is that there is *some* way of taking P and Q and producing the conjunction of P and Q. We can represent this by having a function *And* such that *And*(P, Q) denotes the conjunction of P and Q. To use McCarthy's term, *And*(P,Q) is an *abstract syntax* for representing the conjunction of P and Q. We will represent all the logical operators of the object language by functions in an abstract syntax. [20, p. 79]

The basic strategy here really dates back at least to Tarski (1934) [35], who used exactly this approach in his work on truth definitions. As we noted above, Gödel's (1931) formal representation in PM of syntactic expressions of PM was necessarily more complex, but embodies the same sort of idea. The same approach is used in most modern presentations of formal logic. Applying this idea to our problem, we take $\tau_0(S)$ to be a theory about the formal properties of S. Instead of representing such things as conjunctions of propositional-calculus atoms, we will represent proof trees of (universally closed) predicate-calculus atoms. This raises some minor technical difficulties, to which we now turn.

We have particular interest in the proof trees corresponding to refutations in S. An ordered tree is standardly specified by a pair $\langle A, R\rangle$ where A is a set of vertices and R is a set of sequences of edges (where the edges are ordered pairs of vertices). A labeling then associates a label with each vertex in the tree. We will use a slightly different specification of a labeled tree. A labeled tree will be specified with either a label by itself or a term of the form L/S where L is a label and S is the sequence $[t_1,\ldots,t_n]$ of trees immediately dominated by L. So a label is a tree consisting of just a root node with no arcs. In a tree $l/[t_1,\ldots,t_n]$, l is the label of the root vertex, $t_1,\ldots,t_n$ $(n \geq 0)$ are trees, and an arc connects the root labeled l to the root of each t_i $(0 < i \leq n)$, in order.

Definition. The *set of proof trees over L* is the set of trees with nodes labeled with atoms of L. We will regard these atoms as implicitly universally closed.

Definition. Let G be a goal $\leftarrow A$ where A is an atom. Let Ψ be an SLD refutation $\langle G_0, G_1,\ldots,G_n\rangle$ $(G_0 = G, G_n = \square)$ using input clauses $C_1,\ldots,C_n$ with unifiers $\theta_1,\ldots,\theta_n$, whose composition we will call θ. Then a *proof tree corresponding to* Ψ is a proof tree such that in Ψ, G_{i-1} $(0 < i \leq n)$ is resolved with the head of input clause $C_i = C \leftarrow C_{i_1},\ldots,C_{i_m}$ $(m \geq 0)$ iff there is a node in the tree labeled $C\theta$ with m daughters labeled $C_{i_1}\theta,\ldots,C_{i_m}\theta$. Again, we regard the literals that label the tree as universally closed.

This is a standard notion of a proof tree (e.g., [32]). It is easy to show that for any SLD refutation Ψ of a theory, there is exactly one proof tree corresponding to Ψ. Given any particular proof tree $\mathscr{T}$ and a computation rule R, there is a theory with at least one SLD refutation via R that corresponds to $\mathscr{T}$.

We can now define an interpretation I' with domain D' in terms of our original theory S, its language L, and its interpretation I. The only tricky part is clause (3), because we need to have variables of the object language L in the domain D' of our theory. We regard these variables, which range over D, as universally quantified:

(1) *The domain.* D' is the well-formed expressions of L together with the proof trees over L and sequences of proof trees over L.

(2) *The interpretation of the function symbols.* All function symbols except those in

$$T = \{.^2, [\,]^0, /^2\}$$

receive the Herbrand interpretation. That is, 0-ary functions are mapped

into themselves, and each n-ary ($n > 0$) function f is associated with the mapping from $t_1, \ldots, t_n$ to the function expression $f(t_1, \ldots, t_n)$. The function symbols in T provide representations of (labeled, ordered) trees with more than 0 arcs, as mentioned above. I' associates $/$ with the mapping from l/s to the tree with a root labeled l immediately dominating the subtrees in the sequence s. (We let l/s denote the empty sequence whenever s is not itself a sequence.) I' associates . and [ ] with the standard mappings to sequences:

$$I'(h.t) = \begin{cases} \langle h, t_1, \ldots, t_n \rangle & \text{if } t \text{ is a sequence } \langle t_1, \ldots, t_n \rangle \ (n \geq 0), \\ \langle\,\rangle, & \text{otherwise,} \end{cases}$$

$$I'([\,]) = \langle\,\rangle.$$

(3) *The interpretation of the predicate symbols.* Following standard practice, expressions ϕ of L containing free variables are interpreted relative to an assignment ξ of variables of L to elements of D: $I(\phi)_\xi$ is then defined for all expressions in L. We accordingly define I' in terms of I and assignments ξ. Every n-ary predicate symbol p^n is assigned a set of n-tuples in D' as follows:

$\langle a_1, \ldots, a_n \rangle \in I'(p^n)$

iff

for every ξ, $\quad \langle I(a_1)_\xi, \ldots, I(a_{n-1})_\xi \rangle \in I(p^{n-1})$,

and a_n is a proof tree corresponding to a refutation of $\leftarrow p(a_1, \ldots, a_{n-1})$ in S.

2.3. *Motivating the Interpretation I'*

At this point it would be nice to prove that $I \models S$ iff $I' \models \tau_0(S)$, but unfortunately, this does not hold. Consider the previous example of Section 2.1. It is clear that $I' \not\models \mathit{human}([\,], \mathit{human}([\,])) \leftarrow$, and so $I' \not\models \mathit{human}(x, \mathit{human}(x)) \leftarrow$, and yet this is a clause in $\tau_0(S)$. In short, under our intended interpretation, our transformed theories are false.

We could consider looking for a different interpretation I'' with the property that $I \models S$ iff $I'' \models \tau_0(S)$, but this is obviously not very appealing. We do not *want* an interpretation that verifies $\mathit{human}([\,], \mathit{human}([\,])) \leftarrow$. The most straightforward interpretations I'' with the property $I \models S$ iff $I'' \models \tau_0(S)$ are just not natural.

A more appealing strategy is to define a different transformation τ with the property that $I \models S$ iff $I' \models \tau(S)$. The noted problem with I' arises because the domain we quantify over includes not only the terms of L but also the predications of L and proofs in S that contain such predications. Suppose we formally define a new unary predicate term_L so that it is satisfied by all and only terms of L. Then we could easily define a natural $\tau(S)$ that includes the axioms of this definition and adds a condition to every definite clause in the range of τ_0 requiring that the terms in all but the last arguments of the predicates of that clause be terms of L. Then we would have $I \models S$ iff $I' \models \tau(S)$. However, these transformed theories $\tau(S)$ are more complex than $\tau_0(S)$. Not only does $\tau(S)$ have more symbols than $\tau_0(S)$, but many proofs in $\tau(S)$ would be considerably more complex. Notice that term_L will

have the infinitely many terms of L in its extension: a proper axiomatization would need to make available names for the infinitely many variables of L, and the names of the function expressions will also be infinite if L contains even a single function symbol with positive arity. Consequently, finding a desired instance of a proof using elements of the extension of $term_L$ can be computationally troublesome.

Given the complexity of $\tau(S)$, it is interesting to note the surprising fact that the property that $I' \not\models \tau_0(S)$ does not really matter for our practical purposes. In the first place, it is common for logic programmers to neglect conditions that will properly restrict the provable instances of a goal. Consider for example the standard definition of the *append* relation:

$$append([\,], x, x) \leftarrow$$

$$append([x|y], z, [x|w]) \leftarrow append(y, z, w)$$

When these clauses are added to any theory whose intended domain includes anything other than lists, the first clause is false under the standard interpretation of *append*. These consequences could be ruled out by adding a condition requiring that the arguments of *append* all be lists, but for most purposes this is unnecessary. The same is true for the result of applying τ_0. We should just keep in mind that $\tau_0(S)$ is only an approximation to the correct theory: we must be careful to properly restrict the instances of provable goals. The danger is just that when a logic program contains many such approximations, the programmer may forget the limitations of his axiomatization and get unsound results.

For this reason, it worth noting that we can prove that τ_0 will behave properly on the range of practical cases that we are interested in. Notice that the problematic case $human([\,], human([\,])) \leftarrow$ is not in the range of τ_0, since $human([\,])$ is not an expression in the language L of S—it contains the constant $[\,]$, which is in L' but not in L. If we restrict our attention to refutations of goals that are in the range of τ_0, we will never get an unsound result. So given a theory S and goal G, we do have a sound and efficient method for getting representations of proofs of G in S. The relevant results are the following (the proofs are in Appendix 1):

Proposition 1 (Noninterference). There is an n-step SLD refutation of $S \cup \{G\}$ *using input clauses* $C_1, \ldots, C_n$ *and correct answer substitution* ϵ *iff for some substitution* η *for variables that do not occur in* G, *there is an n-step SLD refutation of* $\tau_0(S) \cup \{\tau_0(G)\}$ *using clauses* $\tau_0(C_1), \ldots, \tau_0(C_n)$ *with correct answer substitution* η.

Corollary 1 (Noninterference). $S \models \forall C$ *iff, for some substitution* η *for variables that do not occur in* C, $\tau_0(S) \models \forall \tau_0(C)\eta$.

Proposition 2 (Representation correctness). Let $G = \leftarrow A$, *where* A *is atomic. There is an n-step SLD refutation of* $\tau_0(S) \cup \{\tau_0(G)\}$ *with a computed answer substitution* η *iff* η *is a substitution* $\{Proof/Tree\}$ *such that Proof is the variable introduced into* $\tau_0(G)$ *by* τ_0, *and Tree is a derivation tree corresponding to an SLD refutation of* $\{G\} \cup S$.

Corollary 2 (Representation correctness). Let $G = \leftarrow A$, *where* A *is atomic, and let* $\tau_0(G) = \leftarrow A'$. *Then* $\tau_0(S) \models \forall A'\eta$ *iff* η, *when restricted to variables in* A', *is a substitution* {*Proof/Tree*} *such that Proof is the variable introduced into* A' *by* τ_0, *and Tree is a derivation tree corresponding to a refutation of* $\{G\} \cup S$.

We conclude this section by considering again the example with which this section began to see that the interpretation we have provided is indeed very close to what was promised. When we prove that

$$\leftarrow man(x, man(x)/[male(x), human(x)])$$

is inconsistent with $\tau_0(S)$, we can conclude that

$$\forall man(x, man(x)/[male(x), human(x)])$$

is entailed by the theory. Keeping in mind that the values of the variable in this term should really be restricted to the terms of the language of S, we have the result that when x is assigned any term in the language of S, $man(x)/[male(x), human(x)]$ is a proof in S. In particular, then, if $x_L, y_L, \ldots$ are variables of the object theory, we can conclude that $man(x_L)/[male(x_L), human(x_L)]$ is a proof in S. That is, $\forall man(x_L)$ can be proven in the object theory by proving $\forall male(x_L)$ and $\forall human(x_L)$, which in turn can be proven directly from unit clauses. This is the interpretation we wanted.

2.4. A Horn-Clause Provability Predicate

A simple extension of our metatheoretic framework will suffice to define a basic "metainterpreter", or object-language provability predicate, for definite-clause theories. Such predicates are used quite frequently [32, 4], and we will want to make use of one when we consider more efficient representations of our problems, below.

We will use the same metatheoretic strategy to define a provability predicate that was used to define proof trees labeled with literals of the object theory. In this case, to represent theories we use sequences whose elements are clauses of the object theory. Introducing the comma as a binary infix functor and the left arrow as a unary postfix and binary infix function symbol, we can represent a sequence of ground clauses, as in

$$[(man(socrates) \leftarrow male(socrates), human(socrates)),$$
$$(male(socrates) \leftarrow),$$
$$(human(socrates) \leftarrow)].$$

Representing a clause with variables is not possible in our system, since we have not provided names for any of the variables. It is not difficult to introduce variable names and explicit quantifiers as in [20], but avoiding names of object-language variables allows us to use simpler metatheories that also have real practical advantages: We can avoid computationally expensive *substitute* and *unify* predicates for object-language expressions. Consequently, rather than representing an object-theory clause with variables, we will use metatheoretic quantification to make claims about all of the instances of certain expressions. So, for example, a

nonground term like the following, occurring in a metatheoretic clause, will have its metatheoretic variables bound in the usual way, and these variables can range over a domain that includes all of the terms—including the variables—of the object theory:

$$[(man(x) \leftarrow male(x), human(x)),$$
$$(male(socrates) \leftarrow), (human(socrates) \leftarrow)].$$

It is clear that this can properly capture the significance of the object theory's variables.[4]

With this representation of definite-clause theories, we can use the following elegant axiomatization of provability:

$$demo(Theory, (P, Q)) \leftarrow demo(Theory, P), demo(Theory, Q)$$
$$demo(Theory, P) \leftarrow member((P \leftarrow), Theory)$$
$$demo(Theory, P) \leftarrow member((P \leftarrow Q), Theory), demo(Theory, Q)$$

These axioms say, roughly, that a conjunction is provable (or *demo*nstrable) from the theory represented by the first argument if both conjuncts are; a literal P is provable if $P \leftarrow$ is an instance of a unit clause in the theory; and P is also provable if there is a clause in the theory of the form $P_0 \leftarrow Q_0$ where there is a substitution θ such that $P_0\theta = P\theta$ and $Q_0\theta$ is provable.

Notice that the axioms defining *demo* have the same flaw as the output of τ_0 and the standard axiomatization of *append*. Strictly speaking, we should require that variables occurring in terms like

$$[(man(x) \leftarrow male(x), human(x)),$$
$$(male(socrates) \leftarrow), (human(socrates) \leftarrow)]$$

range only over object-theory terms. However, we will never be testing any other instances, and so the simple formulation will do for practical purposes. A strictly correct formulation would simply be one in which we add to each of the three axioms the condition that (every instance of) *Theory* is a sequence of object-theory clauses, that Q is a conjunction of one or more positive object theory literals, and that P is a positive object-theory literal.

3. APPLICATIONS

As noted above, there are many applications for theories that produce representations of completed proofs: parsing, justification facilities, and debuggers. We focus on applications with restrictive conditions over the proof representations: conditions that do not preserve the completeness of the proof procedure. We focus in particular on problems that are naturally formulated in terms of overly general axioms together with conditions on proofs which block exactly the derivations of

[4]Nothing very important turns on this decision to avoid variable names. We choose this strategy simply for reasons of expository convenience and efficiency. Note that this strategy does not allow us to treat nonground theories as "first-class objects", and it does not allow us to provide a logical foundations for PROLOG's *assert* and *retract* predicates with variables, as pointed out in [5].

incorrect results. To illustrate the range and power of this approach, we will briefly consider three rather different types of problems that are naturally expressed in this way. We consider basic formulations of these problems before considering more efficient representations in later sections.

3.1. Diagnosis

One recent approach to diagnosis involves finding an inconsistency between the observed behavior of a system and the behavior that follows from a logical specification of its design. Once such an inconsistency is detected, Reiter [27] has shown how we can diagnose the problem by examining the proofs of inconsistency. A *diagnosis* is a minimal set of assumptions whose removal restores consistency. The procedure Reiter defines for computing diagnoses makes use of a relation *tp* that can be regarded as a constraint on proofs in a basic theory of the operation of the system being diagnosed. The relation *tp* is satisfied by inconsistency proofs that do not use certain axioms of the specification of the design and behavior of the system. In this sense, the object-level axioms are overly general in that they define a set of inconsistency proofs, only some of which—those satisfying particular instances of predications involving *tp*—are of interest. Let's consider this in a little more detail.

Since Reiter does not present a Horn-clause formulation, we will adapt one of his examples and show one way of defining his function *tp* as a metatheoretical relation. Consider the circuit shown in Figure 2. The operation of the basic logic gates is easily axiomatized. For each type of gate that might have a fault, we can introduce explicitly a check on the assumption that the gate is operating normally, by conditioning our axioms about the gates as illustrated below:

$$output(Gate,0) \leftarrow and\_gate(Gate), input1(Gate,0), not\_abnormal(Gate)$$

$$output(Gate,0) \leftarrow and\_gate(Gate), input2(Gate,0), not\_abnormal(Gate)$$

$$output(Gate,1) \leftarrow and\_gate(Gate), input1(Gate,1), input2(Gate,1),$$
$$not\_abnormal(Gate)$$

$$output(Gate,1) \leftarrow or\_gate(Gate), input1(Gate,1), not\_abnormal(Gate)$$

$$output(Gate,1) \leftarrow or\_gate(Gate), input2(Gate,1), not\_abnormal(Gate)$$

$$output(Gate,0) \leftarrow or\_gate(Gate), input1(Gate,0), input2(Gate,0),$$
$$not\_abnormal(Gate)$$

$$output(Gate,Out) \leftarrow$$
$$xor\_gate(Gate),$$
$$input1(Gate,In1), input2(Gate,In2), xor(In1,In2,Out),$$
$$not\_abnormal(Gate)$$

$$xor(1,1,0) \leftarrow \quad xor(1,0,1) \leftarrow \quad xor(0,1,1) \leftarrow \quad xor(0,0,0) \leftarrow$$

$$disjoint(0,1) \leftarrow \quad disjoint(1,0) \leftarrow$$

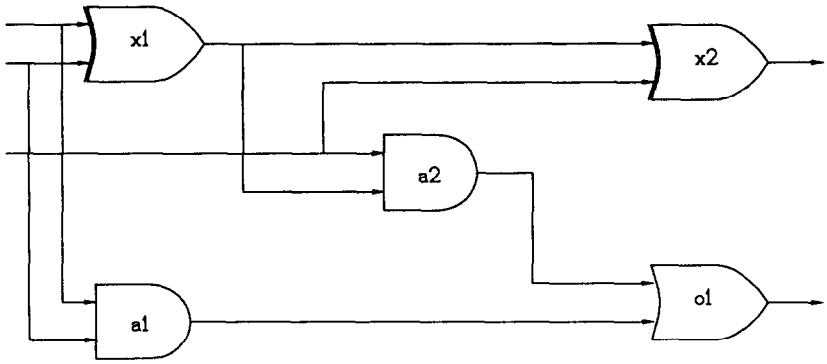

FIGURE 2. A full adder.

To specify the design of the particular circuit we must specify the gates of the circuit:

$$and\_gate(a1) \leftarrow \quad and\_gate(a2) \leftarrow \quad xor\_gate(x1) \leftarrow$$
$$xor\_gate(x2) \leftarrow \quad or\_gate(o1) \leftarrow$$

We add our assumptions that these gates are operating normally:

$$not\_abnormal(a1) \leftarrow \quad not\_abnormal(a2) \leftarrow \quad not\_abnormal(x1) \leftarrow$$
$$not\_abnormal(x2) \leftarrow \quad not\_abnormal(o1) \leftarrow$$

And finally we must specify how the gates are connected in such a way that every needed value can be determined from the specified inputs to the circuit. For this purpose the following conditionals suffice (although the corresponding biconditionals would of course be true):

$$input1(x2, Value) \leftarrow output(x1, Value)$$
$$input2(x2, Value) \leftarrow input1(a2, Value)$$
$$input2(a2, Value) \leftarrow output(x1, Value)$$
$$input1(a1, Value) \leftarrow input1(x1, Value)$$
$$input2(a1, Value) \leftarrow input2(x1, Value)$$
$$input1(o1, Value) \leftarrow output(a2, Value)$$
$$input2(o1, Value) \leftarrow output(a1, Value)$$

Observed behavior then can be represented with axioms like the following:

$$input1(x1, 1) \leftarrow \quad input2(x1, 0) \leftarrow \quad input1(a2, 1) \leftarrow$$
$$output(x2, 1) \leftarrow \quad output(o1, 0) \leftarrow$$

This is a complete, basic theory of the operation of our faulty circuit.

Given a theory like this, we want to check for an inconsistency between the observed output and the output that the design specification determines. This can be done by adding the Horn-clause expression of the fact that no output gate can

have two values X, Y that are disjoint—a "goal":

$\leftarrow output(Gate, X), disjoint(X, Y), output(Gate, Y)$

If we can refute this negative clause (showing the inconsistency of the theory comprising this clause together with the other axioms), then we know the circuit is faulty. For convenience, we will use the equivalent strategy of adding the following clauses and proving the inconsistency of the result:

$inconsistency \leftarrow output(Gate, X), disjoint(X, Y), output(Gate, Y)$

$\leftarrow inconsistency$

This alternative strategy, using a negative *unit* clause, will allow our metatheoretic transformation to give us a single proof tree to inspect—a proof tree with *inconsistency* at the root.

Since the questionable assumptions of the theory have been marked as assumptions that the components are *not_abnormal*, Reiter defines a diagnosis to be a minimal set C of components such that removing *not_abnormal(Component)* for every *Component* in C restores consistency. In this example, there are three diagnoses: $[x1]$, $[x2, a2]$, $[x2, o1]$. These diagnoses can be determined by examining the proofs of inconsistency to see which *not_abnormal* assumptions about the circuit were used. This is exactly what τ_0 allows us to do. Applying τ_0 to the axioms presented above, we produce a new 1-place predicate again named *inconsistency* with the property that the provable instances of *inconsistency(Proof)* are exactly those in which *Proof* is instantiated to a proof of *inconsistency* in the original theory. This transformed theory can then be embedded in a larger metatheory which defines special properties that we want these proofs to have.

Since the questionable assumptions of the theory have been marked as assumptions that the components are *not_abnormal*, we can examine our proofs to find which components' operations were assumed to be normal. We simply check the proof tree for nodes labeled *not_abnormal(Component)*. The set of components named in these nodes in a particular proof is what Reiter calls a *conflict set*. One way to compute the diagnoses involves finding every conflict set by finding every proof of inconsistency, and then computing the possible diagnoses from the set of conflict sets. Clearly this approach is not feasible if there is a very large or infinite set of inconsistency proofs, or if the proofs are very complex.

Reiter defines a much more efficient way of computing a diagnosis, though, which does not, in general, require finding all proofs of inconsistency. This procedure uses a procedure *tp* which searches for inconsistency proofs in which various *not_abnormal* assumptions are not used, and returns a conflict set of a proof if one is found. Roughly, the idea is to find one conflict set and then efficiently pare it down to a minimal conflict set by calling *tp* to see if there are inconsistency proofs using only some proper subsets of the conflict sets already found. (See Reiter [27] for the details of this "paring-down" algorithm.) Notice that *tp* is exactly the sort of relation that is easily represented with our metatheoretic approach. We can define *tp* as a two-place relation between a list of components and a conflict set for a proof, where the proof is one that does not use assumptions about the components listed in the first argument.[5] If there is no proof that does not make assumptions about the components listed in the first argument to this predicate, the second argument is defined to be the special

expression *consistency*. The following axioms suffice:

$$tp(Components, ConflictSet) \leftarrow$$
$$\quad inconsistency(Proof),$$
$$\quad \neg\, uses\_components(Components, Proof),$$
$$\quad conflict\_set(Proof, ConflictSet)$$
$$tp(Components, consistency) \leftarrow$$
$$\quad \neg\,(\, inconsistency(Proof),$$
$$\qquad \neg\, uses\_components(Components, Proof)\,)$$

This axiomatization of *tp* is correct, but notice that it is inefficient for left-to-right theorem provers: it finds a complete *Proof* before checking to make sure that the *Proof* does not use any of the components named in the list *Components*. This inefficient generate-and-test solution procedure can be transformed with the techniques described below to yield a provably correct theory from which diagnoses can be established using a minimal number of constrained proofs.

The "paring-down" procedure Reiter defines to call *tp* in the computation of diagnoses is also easily represented in Horn clauses, and can simply be added to the metatheory (though this is not relevant to the present material, and so will be left to the reader). Appropriate instances of *tp* can also be used to efficiently find diagnoses that assume less than *n* faults. The flexibility of our metatheoretical approach makes other elaborations of Reiter's basic approach straightforward. For example, in some types of systems, there may be pairs of components which very rarely fail together. In such cases, it would be easy to tailor the search to find any diagnoses that remain without the assumption that both members of any such pair have failed.

3.2. Planning

Planning problems can be expressed in terms of an appropriate relation between a plan and a goal such that performing the plan will achieve the goal. In some formulations of planning problems, the plan itself is closely related to a proof. D. H. D. Warren noted this point in discussing WARPLAN, for example: "... there is a one-to-one correspondence between plans and the proofs that these plans achieve the desired goals. ... Thus the terms representing plans can equally well be said to represent *proofs* of these plans. ..." [38]. On this approach, actions are expressed by rules, so that an action is a way of making certain propositions true.

In a recent similar project, Bibel has also used this correspondence between proofs and plans to provide an explicit formulation of planning as a problem of finding a proof satisfying certain constraints [1]. He shows that "connection

[5]Rather than blocking the use of certain assumptions, Reiter gives a theorem prover sets of clauses from which those assumptions are absent. This requires that the system that calls the theorem prover have a (metatheoretic) way of naming the clauses used by the theorem prover. We have introduced such an extension of our framework in defining the *demo* predicate, but the simple approach described here does not require that extension.

proofs" meeting certain constraints correspond to plans. Recasting Bibel's constraints in a way appropriate for resolution proofs in Horn-clause theories is not trivial.[6] Bibel adopts the simplifying assumption that every proposition whose truth is affected by an action is listed in the antecedent of the action rule. Then his constraint is roughly that, in a forward chaining system, the axioms used to establish the preconditions of an action should not be assumed to hold later in the proof, "after the action".[7]

A simple example will illustrate the correspondence between plans and proofs meeting certain constraints. It will also illustrate some minor difficulties with representing planning problems in a Horn-clause resolution system. Consider a simple "blocks world" example with blocks a and b. We can represent an initial state as follows:

$$clear(b) \leftarrow$$

$$on(b, a) \leftarrow$$

$$on(a, floor) \leftarrow$$

To represent a simple *move* action, we introduce a predicate that indicates what was moved, what it was moved from, and what is was moved to: *move(Block, OldSupport, NewSupport).* Then we can represent the acceptable preconditions of the action with clauses like the following:

$$move(Block, OldSupport, floor) \leftarrow$$
$$on(Block, OldSupport),$$
$$\neg\, identical(OldSupport, floor),$$
$$clear(Block)$$
$$move(Block, OldSupport, Support) \leftarrow$$
$$clear(Support),$$
$$on(Block, OldSupport),$$
$$\neg\, identical(Block, Support),$$
$$clear(Block)$$

We can represent the effects of a *move* with a sentence like the following:

$$move(Block, OldSupport, Support) \rightarrow on(Block, Support) \wedge clear(OldSupport)$$

This can be expressed as two Horn clauses, but then it should be kept in mind that

[6]Bibel underestimates the difficulty of representing his approach in Horn-clause, resolution-based approaches. He suggests that the "linearity" constraint used for connection proofs could be applied directly in Horn-clause resolution proofs [1, p. 129], but this idea is mistaken, as shown by his own Figure 1, in which the resolution proof corresponding to a simple plan is shown to lack "linearity".

[7]The reason I focus on the planning strategies of Warren and Bibel is that they are so directly expressible in our framework. Both use the proof context to indicate the notions of "before" and "after" the action. This rather confusing and inelegant feature of their approach is avoided in other formulations—see especially Kowalski's [15]. These more elegant formulations are also naturally expressed in metatheoretic terms, as noted by Kowalski [15, p. 136], and with a little effort can also be represented with a formulation similar to that used here.

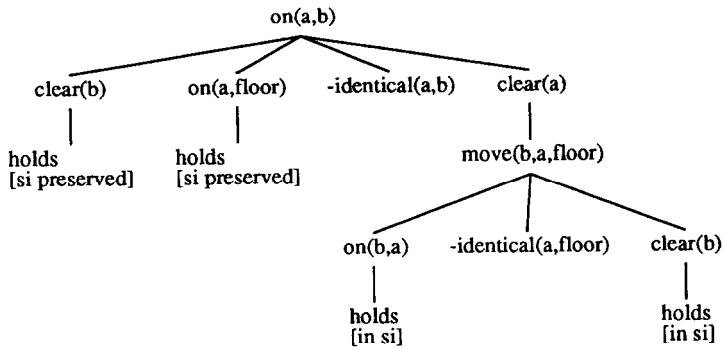

FIGURE 3. A planning-problem proof tree.

one *move* action suffices to make both consequents true:

$on(Block, Support) \leftarrow move(Block, OldSupport, Support)$

$clear(OldSupport) \leftarrow move(Block, OldSupport, Support)$

A minor problem is raised by this Horn reformulation of the clause for *move*. An action typically makes several things true, but in a Horn-clause system we must use different clauses to establish each of the different (atomic) conclusions. Yet this will remove the one-to-one correspondence between plans and proofs, and it also introduces inefficiency in having to find "proofs" of the same action more than once. For these reasons, we can add rules for *on* and *clear* that say they already hold:

$on(Block, Support) \leftarrow holds$

$clear(OldSupport) \leftarrow holds$

And then we put a constraint on the occurrence of *holds* nodes in the proof to the effect that the parent is made true by an action that has already been performed. More precisely, *holds* nodes are allowed only if their parents are preserved by earlier actions and true in the initial state s_i, or if they are established by an earlier action and preserved thereafter. (In the tree displayed in Figure 3, we represent the former justification of the *holds* relation with the annotation "in s_i" when there have been no earlier actions and with "s_i preserved" when the earlier actions have preserved the relevant features of the initial state s_i.)

Most Horn-clause resolution systems reason backward from the goal (i.e. from a negative clause). That means that the needed actions will be found in reverse order, and the preconditions for the actions immediately prior to the final state may have been established not in the initial state but by previous actions. When we speak of an "earlier" or "previous" action we mean one that occurs later in the proof. The conditions on *holds* nodes will be enforced with respect to nodes occurring later in the proof.

Applying τ_0 to this theory, we can establish correct representations of its proofs, and some of the proofs do correspond to valid plans. For example, a proof tree for $\leftarrow on(a, b)$ is shown in Figure 3, with annotations on the *holds* nodes to indicate their justifications. This proof corresponds to the following plan for

achieving $on(a, b)$ from the initial state $clear(b), on(b, a), on(a, floor)$:

$move(b, a, floor); move(a, floor, b).$

In effect, it is proofs of this kind that are found by Warren's and Bibel's systems.

With a correct formulation of the *holds_justified(Proof)* constraints on the occurrences of *holds* in a given *Proof* we could formulate our planning problems as follows: Given an atomic goal G and the basic theory of the domain S, obtain a proof tree *Proof* that satisfies *holds_justified(Proof)* by finding a suitable refutation of $\tau_0(G) \cup \tau_0(S)$. The sequence of actions can then be collected in the reverse order of their preorder occurrence in the tree. The plan corresponding to the proof tree displayed in Figure 3 can thus be obtained as an instance of *plan(Plan)* provable in a theory containing the result of applying τ_0 to our basic theory and the following clause:

$$plan(Plan) \leftarrow on(a, b, Proof),$$
$$holds\_justified(Proof),$$
$$plan\_in\_proof(Proof, Plan)$$

While this representation of the problem is correct, it is not feasible. Once again, the solution strategy with a left-to-right theorem prover is, in effect, the terribly inefficient "generate-and-test" approach. Entailments of the original, overly general theory are generated and then tested against the constraints. Since in many problems the constraints rule out *most* of the generated cases, it would be much more efficient to apply the tests as soon as possible, to partial proofs, rather than generating complete proofs before testing them. We will consider how to formulate a logical representation of the problem that can be more efficiently used, after considering some other applications of our technique.

Another problem with our representation of plans is that the proofs specify a total ordering of the steps of the plan, backtracking to find alternative orderings when necessary. A metatheoretic approach which uses more efficient partial orderings of subplans (as in [7, 22]) is under development.

3.3. Natural-Language Parsing

A division of labor between levels of representation has been used, if only implicitly, in many approaches to parsing. Many parsers can be seen, at some level of abstraction, as using rewrite rules together with additional principles which "filter" out some of the derivations allowed by the rewrite rules. This "filtering" is done partly by the action of "agreement rules" and similar principles whose domain is relatively restricted.[8] The more challenging job for the parser is to properly enforce conditions that are not naturally stated as requirements on sister nodes, conditions that depend on larger portions of the structural representation. These conditions have prompted a great variety of special and complex parsing

[8]For example, most PROLOG parsers for fragments of natural language fit this description. An appendix of [24], for example, presents a DCG recognizer with tests. Without the tests, the DCG corresponds to an overly general grammar. The tests, in effect, filter out the unwanted derivations from the overly general grammar.

mechanisms. Our approach allows a simple and elegant logical representation of such conditions to be used directly.

A simple example suffices to illustrate the metalogical approach to parsing.[9] Pereira and Warren [24] define an elegant and efficient "definite-clause grammar" (DCG) representation of context free grammars. Strings in the extensions of nonterminal categories are represented with "difference lists"—pairs of lists $L0, L$ that represent the string consisting of the elements of L after the elements of $L0$ have been removed from the tail of L. With this representation, there is a simple "translation" from CFGs to definite clauses (DCs) that can be efficiently used with "left-to-right" Horn-clause proof techniques. The context-free derivation trees for the grammar correspond to logical derivation or proof trees in the definite-clause representation, and so it is natural to attempt to represent constraints on acceptable structural representations as constraints on logical derivations.

A definite-clause grammar can easily be extended to enforce feature agreement. The following definite-clause grammar with features will suffice to illustrate the use of Chomskian constraints:

$$s(L0, L) \leftarrow np(F, Index, L0, L1), vp(L1, L)$$
$$np(-wh, Index, L0, L) \leftarrow name(L0, L)$$
$$np(-wh, Index, L0, L) \leftarrow det(L0, L1), n(L1, L)$$
$$np(-wh, Index, L0, L) \leftarrow det(L0, L1), n(L1, L2), sbar(L2, L)$$
$$np(F, Index, L0, L) \leftarrow trace(Index, L0, L)$$
$$np(+wh, Index, L0, L) \leftarrow rel\_pro(L0, L)$$
$$trace(Index, L, L) \leftarrow$$
$$sbar(L0, L) \leftarrow comp(L0, L1), s(L1, L)$$
$$comp(L0, L) \leftarrow np(+wh, Index, L0, L)$$
$$vp(L0, L) \leftarrow verb(L0, L1), np(F, Index, L1, L)$$
$$name([mary|L], L) \leftarrow$$
$$det([the|L], L) \leftarrow$$
$$n([man|L], L) \leftarrow$$
$$rel\_pro([who|L], L) \leftarrow$$
$$verb([likes|L], L) \leftarrow$$

In this simple grammar, special subclasses of *np* are singled out by the $+wh$ and $-wh$ features. Relative pronouns like "who" are $+wh$; phrases like "the man" and "the man who mary likes" are $-wh$; and the special category *trace*, which always expands to the empty string, is the only noun phrase that can be either $+wh$ or $-wh$. Also, notice that every *np* has an uninstantiated (and hence universally quantified) argument, *Index*, which will play a special role in enforcement of the constraints.

[9] This example is treated in detail in [33].

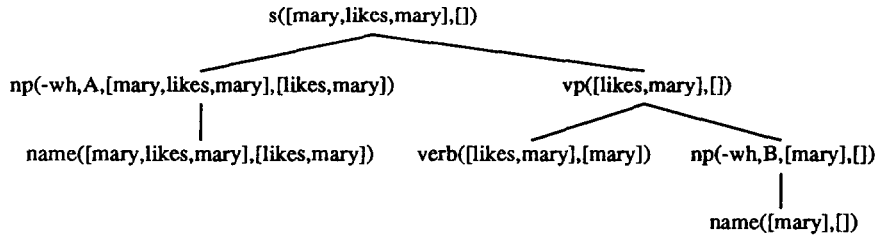

FIGURE 4. A proof tree for $\leftarrow s([mary, likes, mary], [\,])$.

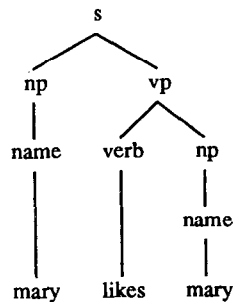

FIGURE 5. A parse tree for the sentence "mary likes mary".

According to this theory, the set of sentences includes "mary likes the man", "the man who mary likes likes mary", "the man who likes mary likes mary", "the man who likes the man who likes mary likes mary", and so on. Notice, though, that the set also includes "likes", "likes mary", "likes the man likes", and other strings that are not English sentences. We can eliminate the acceptance of these latter strings by imposing constraints on derivations. The similarity between grammatical derivation trees and proof trees from a definite-clause representation of the grammar is clear.[10] Consider the proof tree of Figure 4. This corresponds to a parse tree that would usually be displayed in the form shown in Figure 5. In the remainder of this section we will display the more readable parse trees as abbreviations for the corresponding proof trees, and constraints on these proof trees will be used to block the derivation of ungrammatical strings.

For purposes of illustration, we will use three constraints that are simplified versions of constraints actually proposed in government-binding theory.[11] Our first constraint is a simplistic rendering of the "θ-criterion": an *np* that is immediately dominated by *comp* is in an $\bar{A}$-position, and every *np* must either be in A-position or else coindexed with exactly one *np* that is in A-position [37]. The simpler condition we want to impose here can be expressed in the following way:

1. *A theta condition.* If *comp* immediately dominates an *np*, that *np* must be coindexed with a *trace* that is not dominated by *comp*.

[10]This correspondence, suggested in [24], is precisely specified in [34].

[11]A formalization of these constraints that is not simplified is presented complete detail in [34].